

# 1-1: Introduction to Recommender Systems

Introduction to Recommender Systems

# Today's Learning Objectives

- Understand what a recommender system is
- Some history and background
- NOT: course details (next lecture)

# A Bit of History

- Ants, Cavemen, and Early Recommender Systems
  - The emergence of critics
- Information Retrieval and Filtering
- Manual Collaborative Filtering
- Automated Collaborative Filtering
- The Commercial Era

# A Bit of History

- Ants, Cavemen, and Early Recommender Systems
  - The emergence of critics
- Information Retrieval and Filtering
- Manual Collaborative Filtering
- Automated Collaborative Filtering
- The Commercial Era

# Information Retrieval

- Static content base
  - Invest time in indexing content
- Dynamic information need
  - Queries presented in “real time”
- Common approach: TFIDF
  - Rank documents by term overlap
  - Rank terms by frequency

# Information Filtering

- Reverse assumptions from IR
  - Static information need
  - Dynamic content base
- Invest effort in modeling user need
  - Hand-created “profile”
  - Machine learned profile
  - Feedback/updates
- Pass new content through filters

# A Bit of History

- Ants, Cavemen, and Early Recommender Systems
  - The emergence of critics
- Information Retrieval and Filtering
- Manual Collaborative Filtering
- Automated Collaborative Filtering
- The Commercial Era

# Collaborative Filtering

- Premise
  - Information needs more complex than keywords or topics: quality and taste
- Small Community: Manual
  - Tapestry – database of content & comments
  - Active CF – easy mechanisms for forwarding content to relevant readers

# A Bit of History

- Ants, Cavemen, and Early Recommender Systems
  - The emergence of critics
- Information Retrieval and Filtering
- Manual Collaborative Filtering
- Automated Collaborative Filtering
- The Commercial Era

# Automated CF

- The GroupLens Project (CSCW '94)
  - ACF for Usenet News
    - users rate items
    - users are correlated with other users
    - personal predictions for unrated items
  - Nearest-Neighbor Approach
    - find people with history of agreement
    - assume stable tastes

xrn - version 8.01-beta-3

|                          |       |                 |
|--------------------------|-------|-----------------|
| + 17107 Cream Brulee     | ##### | [51] Art Poe    |
| 17108 Mop Sauce          | ###   | [33] Art Poe    |
| 17109 Banana Cream Pie   | NA    | [55] Art Poe    |
| 17110 Baked Potato Soup  | ##### | [45] Art Poe    |
| 17111 %Minestrone        | ###   | [53] Sam Waring |
| 17112 %Apple Pandowdy    | ####  | [52] Sam Waring |
| 17113 %Lebkuchen         | ####  | [64] Sam Waring |
| 17114 %Plum Chutney      | ##### | [53] Sam Waring |
| 17115 %Swedish Flatbread | ##### | [45] Sam Waring |

Operations apply to current selection or cursor position

Quit Next unread Next Prev Catch up Post Gripe Next group

From: "Art Poe" <apoe@unicom.net>  
Subject: Cream Brulee  
Organization: Unicom

MasterCook export: Cream Brulee

\* Exported from MasterCook \*

Creme Brulee

Recipe By : Food and Wine - Dec85  
Serving Size : 8 Preparation Time :0:00  
Categories : Desserts

| Amount | Measure     | Ingredient -- Preparation Method |
|--------|-------------|----------------------------------|
| 4      | Cups        | Heavy Cream                      |
| 1      | Pinch       | Vanilla bean                     |
| 8      |             | Salt                             |
| 3/4    | Cup         | Egg yolks                        |
| 2      | Tablespoons | Sugar                            |
| 8      | Tablespoons | Brown sugar                      |

Preheat the oven to 300F. In a heavy medium saucepan, combine the cream, vanilla bean and salt. Warm over moderate heat until the surface begins to shimmer, about 5 minutes. In a large bowl, stir the egg yolks and sugar until blended. Pour in the hot cream and stir gently to avoid forming air bubbles. Strain the custard into a large measuring glass and skim off any surface air bubbles. (Rinse the vanilla bean and reserve for future use.) Place 8 3/4-cup ramekins in a roasting pan. Pour the custard into the ramekins, filling them up to the rim. Place the roasting pan in the oven and pour in enough warm water to reach halfway up the sides of the ramekins. Cover loosely with foil and bake for 1 1/4 hours, or until the custard is firm around the edges. (It may still be wobbly in the center but it will firm up as it chills.) Remove the ramekins from the

This article is great, I'd like to see more like this one!

Save Reply Forward Followup Followup & Reply Cancel Rot-13 Translate

Toggle header Print artRate1 artRate2 artRate3 artRate4 artRate5

# Does it Work?

- Yes: The numbers don't lie!
  - Usenet trial: rating/prediction correlation
    - rec.humor: 0.62 (personalized) vs. 0.49 (avg.)
    - comp.os.linux.system: 0.55 (pers.) vs. 0.41 (avg.)
    - rec.food.recipes: 0.33 (pers.) vs. 0.05 (avg.)
  - Significantly more accurate than predicting average or modal rating.
  - Higher accuracy when partitioned by newsgroup

# It Works Meaningfully Well!

- Relationship with User Behavior
  - Twice as likely to read 4/5 than 1/2/3
- Users *Like* GroupLens
  - Some users stayed 12 months after the trial!

# A Bit of History

- Ants, Cavemen, and Early Recommender Systems
  - The emergence of critics
- Information Retrieval and Filtering
- Manual Collaborative Filtering
- Automated Collaborative Filtering
- The Commercial Era

# Am

Amazon.com: Recommended for You - Mozilla Firefox

File Edit View History Bookmarks Yahoo! Tools Help

jkonstan

https://www.amazon.com/gp/yourstore/home/102-9592175-0760921?ie=UTF8&p... Google

Google Search Check AutoLink Subscribe AutoFill Options

amazon.com Joseph's Amazon.com See all 41 Product Categories Your Account Cart Your Lists Help

Your Browsing History Recommended For You Rate These Items Improve Your Recommendations Your Profile Learn More

Search Amazon.com Go Find Gifts Web Search Go

Joseph, Welcome to Your Amazon.com™ (If you're not Joseph A. Konstan, [click here.](#))

Today's Recommendations For You

1 2 3 4 5

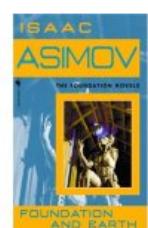
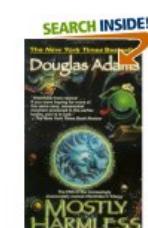
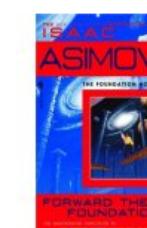
Here's a daily sample of items recommended for you. Click here to [see all recommendations.](#)

[Netgear FS105NA ProSafe 5 Port 10 / 100Mbps Des...](#) [Choosing a Jewish Life \(Paperback\)](#) [Foundation's Fear \(Mass Market Paperback\)](#) [Black Identities \(Paperback\)](#)

African-American Studies All Categories Broadway & Vocalists Business Classic Vocalists Classics  
Contemporary Cultural Dilbert Drama Fantasy Jazz Look Inside Art Books Look Inside  
**Entertainment Books** Look Inside History Books Look Inside Nonfiction Books  
Look Inside Science Fiction & Fantasy Books Management Pop Science Fiction  
Shakespeare, William Study Traditional Pop Traditional Vocal Pop United States Urban

Complete Your Series

[Foundation and Earth](#) Mass Market Paperback by Isaac Asimov  
[Mostly Harmless](#) Mass Market Paperback by Douglas Adams  
[Forward the Foundation](#) (Foundation Novels) Mass Market Paperback by Isaac Asimov

Your Recent Shopping  
[Recently Viewed Items \(0\)](#)  
[Your Shopping Cart \(0\)](#)  
[Open & Recently Shipped Orders](#)

Your Lists  
[Your Wish List](#)  
[Your Gift List](#)  
[Your Shopping List](#)

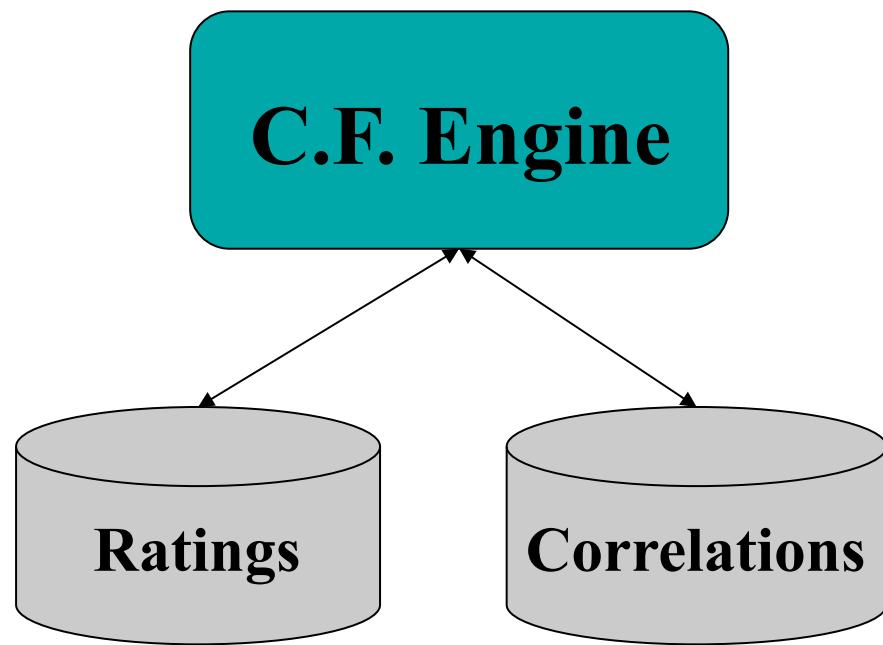
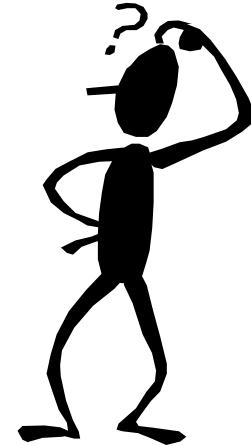
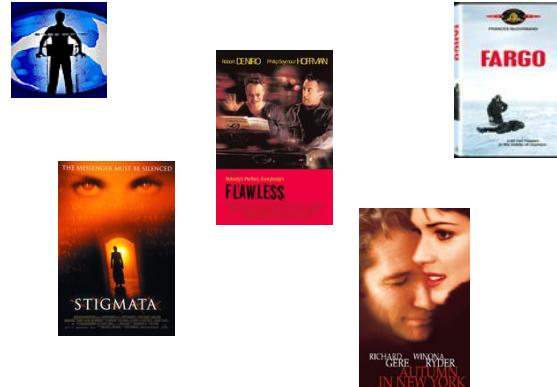
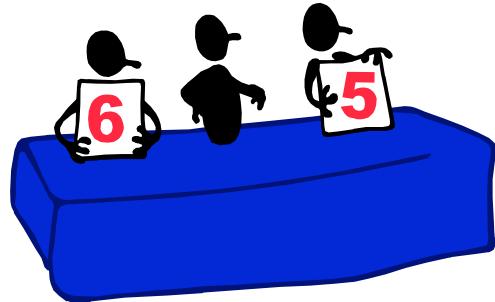
Your Community  
[Your Amazon Friends](#)  
[Your Interesting People](#)  
[Your Reminders](#)

Done www.amazon.com

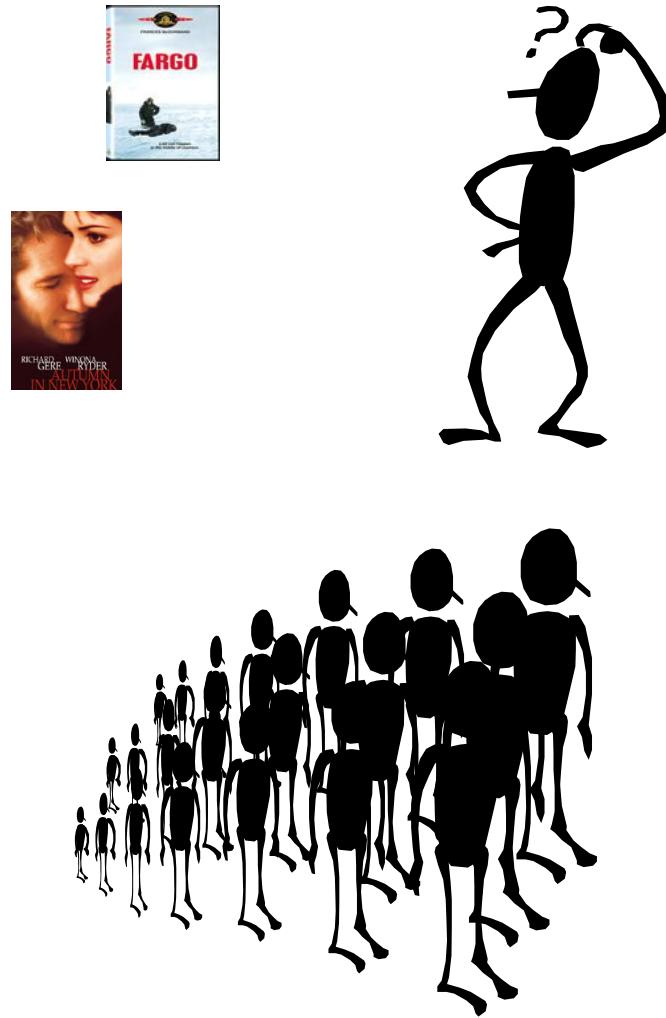
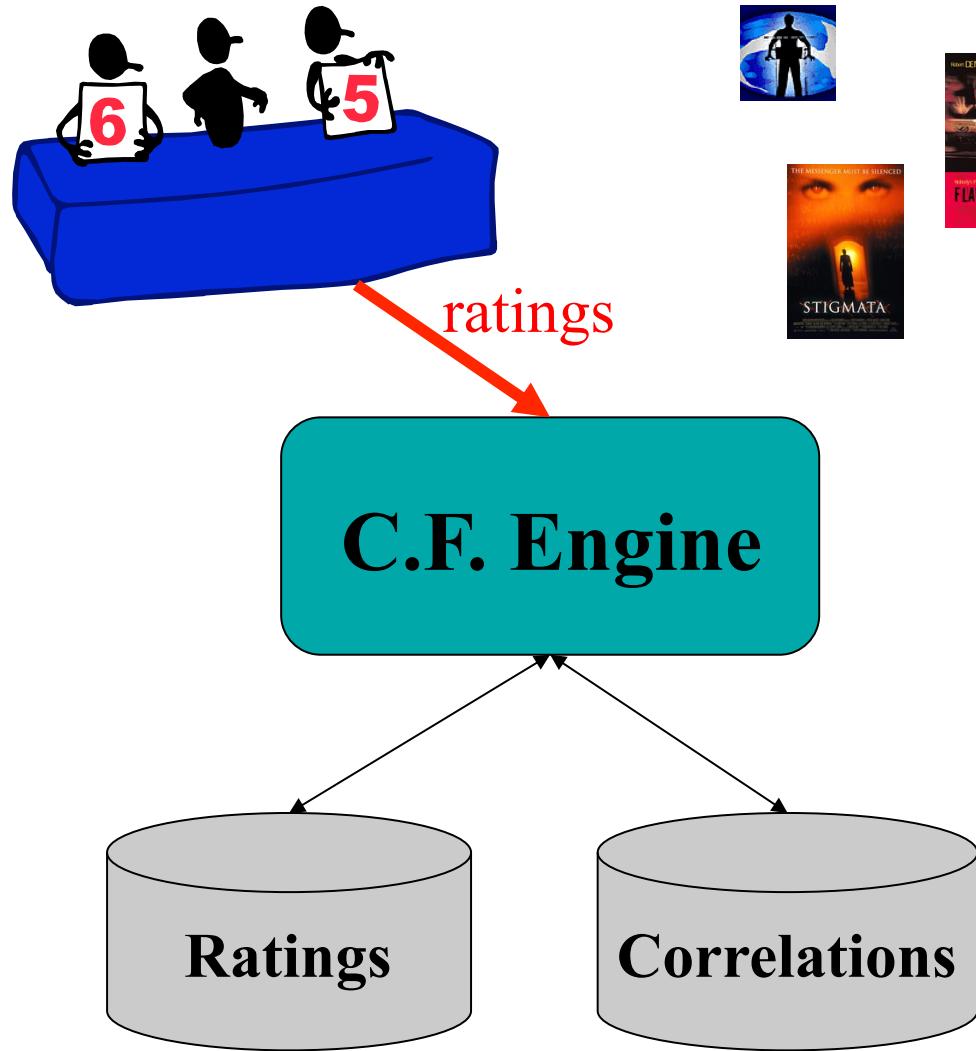
# **A TOUR OF MOVIELENS (CLASSIC COLLABORATIVE FILTERING)**

Introduction to Recommender Systems

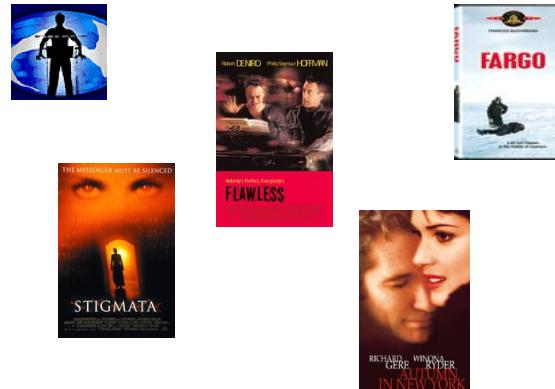
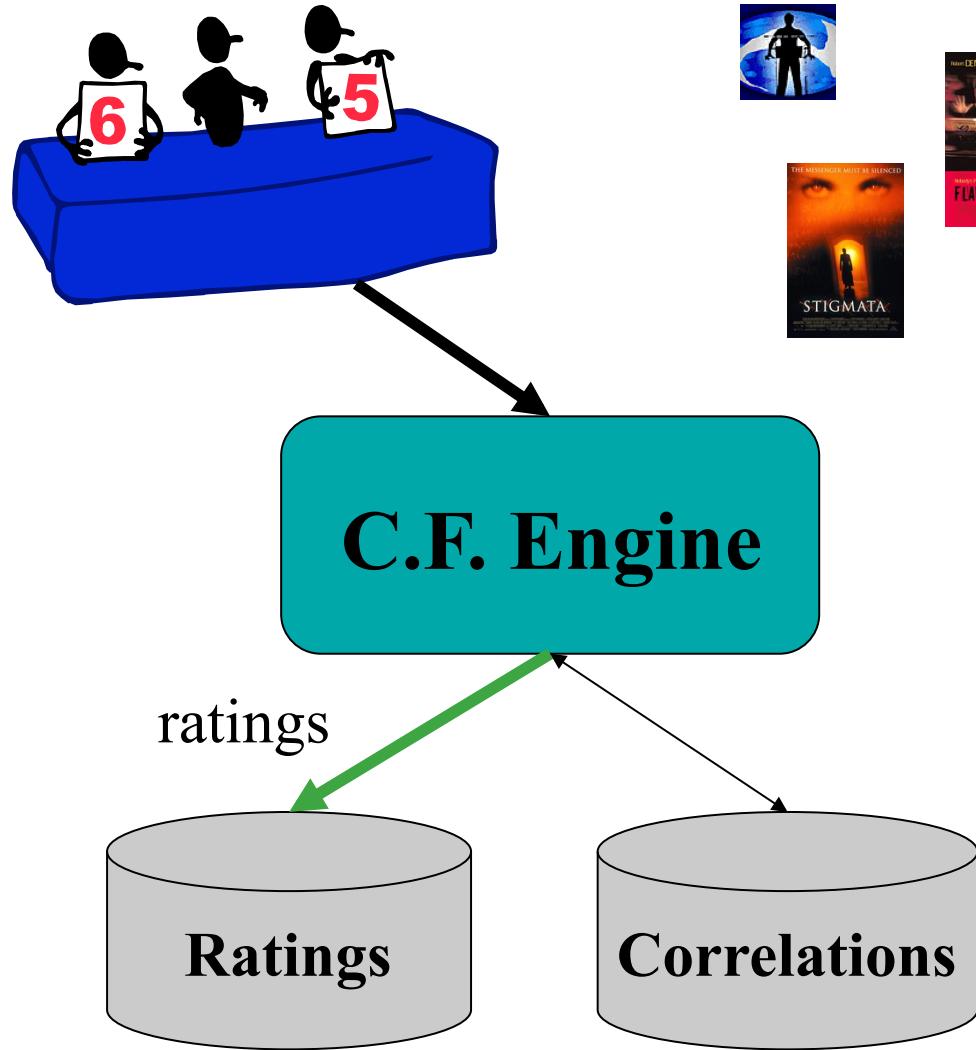
# K-Nearest Neighbor User-User



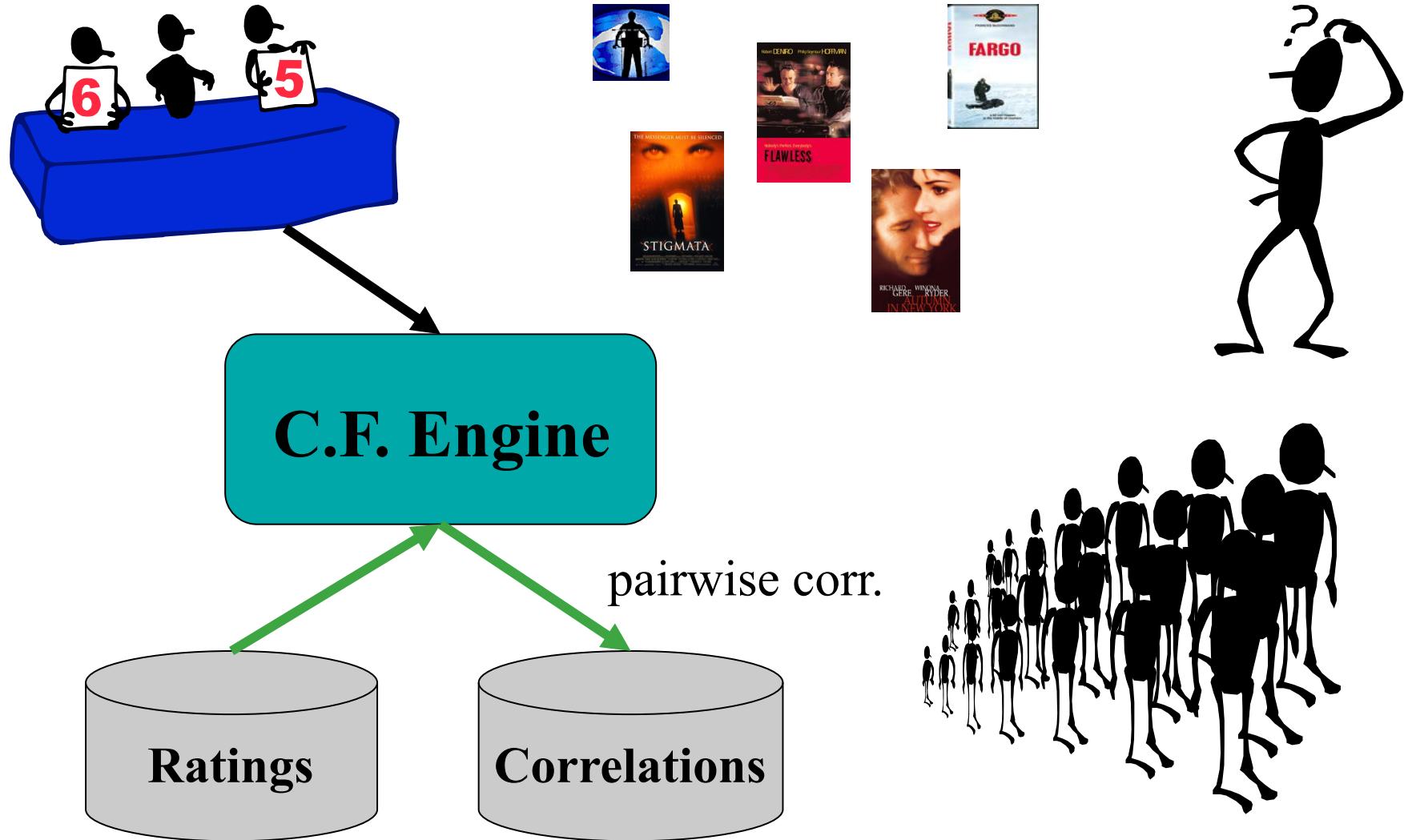
# CF Classic: Submit Ratings



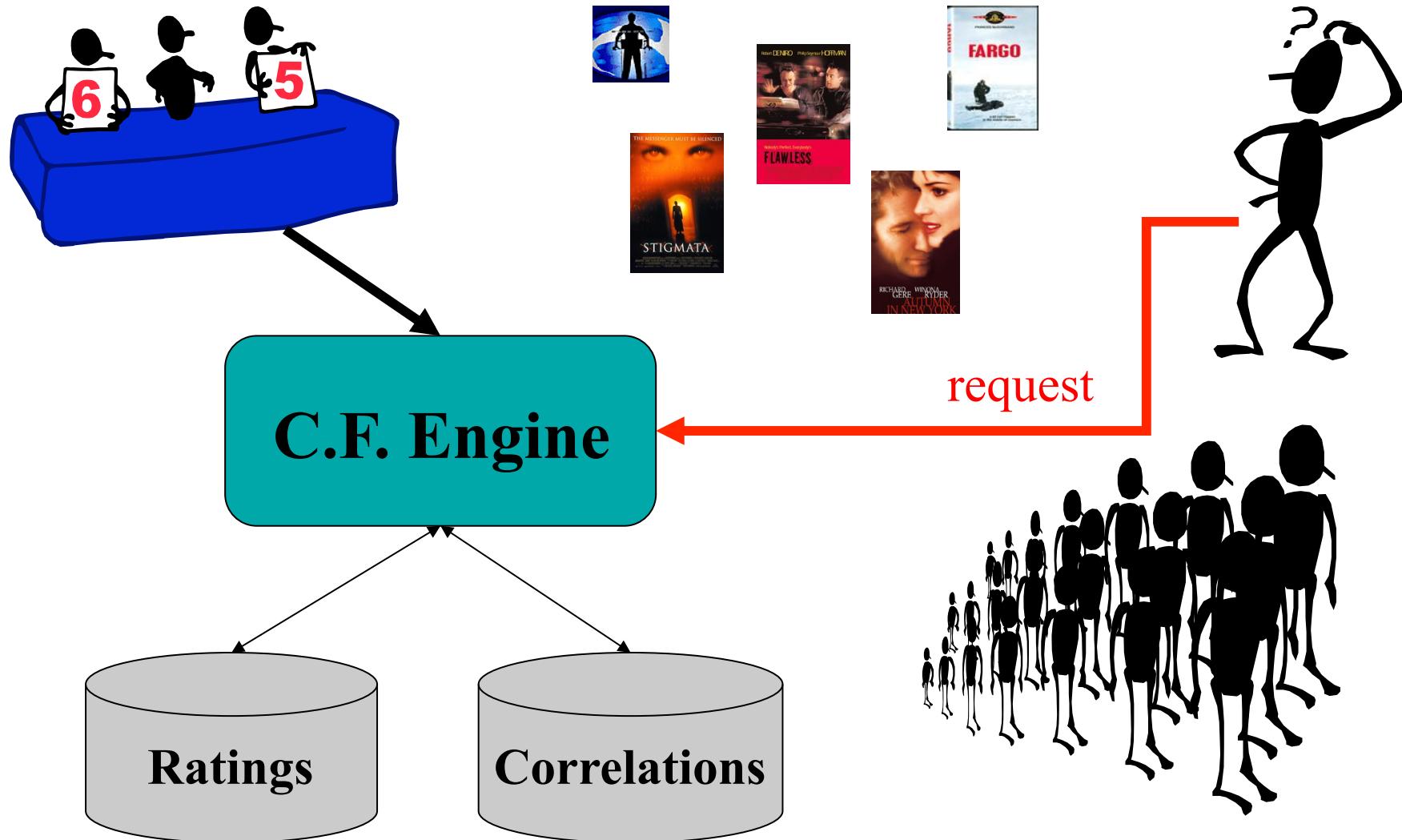
# CF Classic: Store Ratings



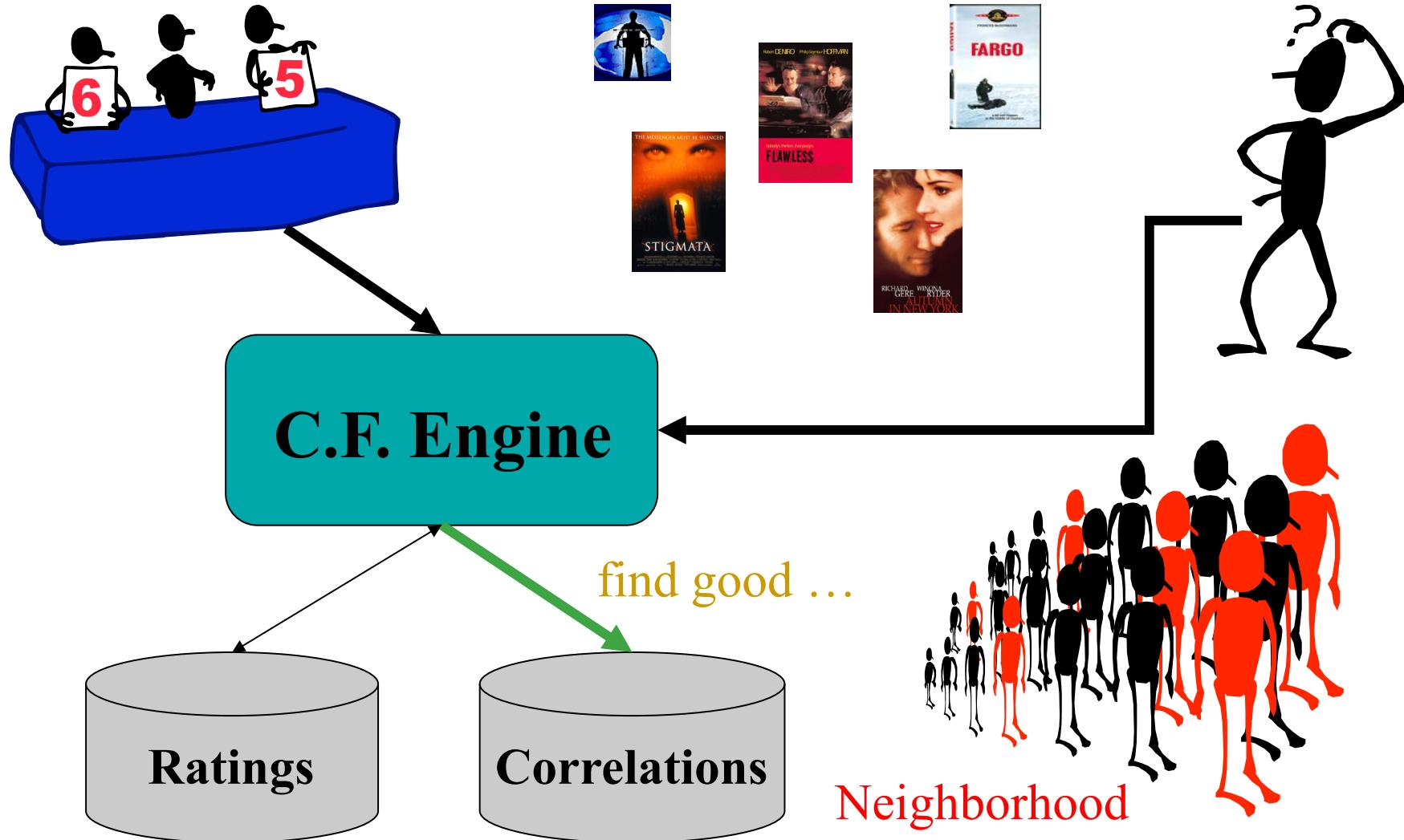
# CF Classic: Compute Correlations



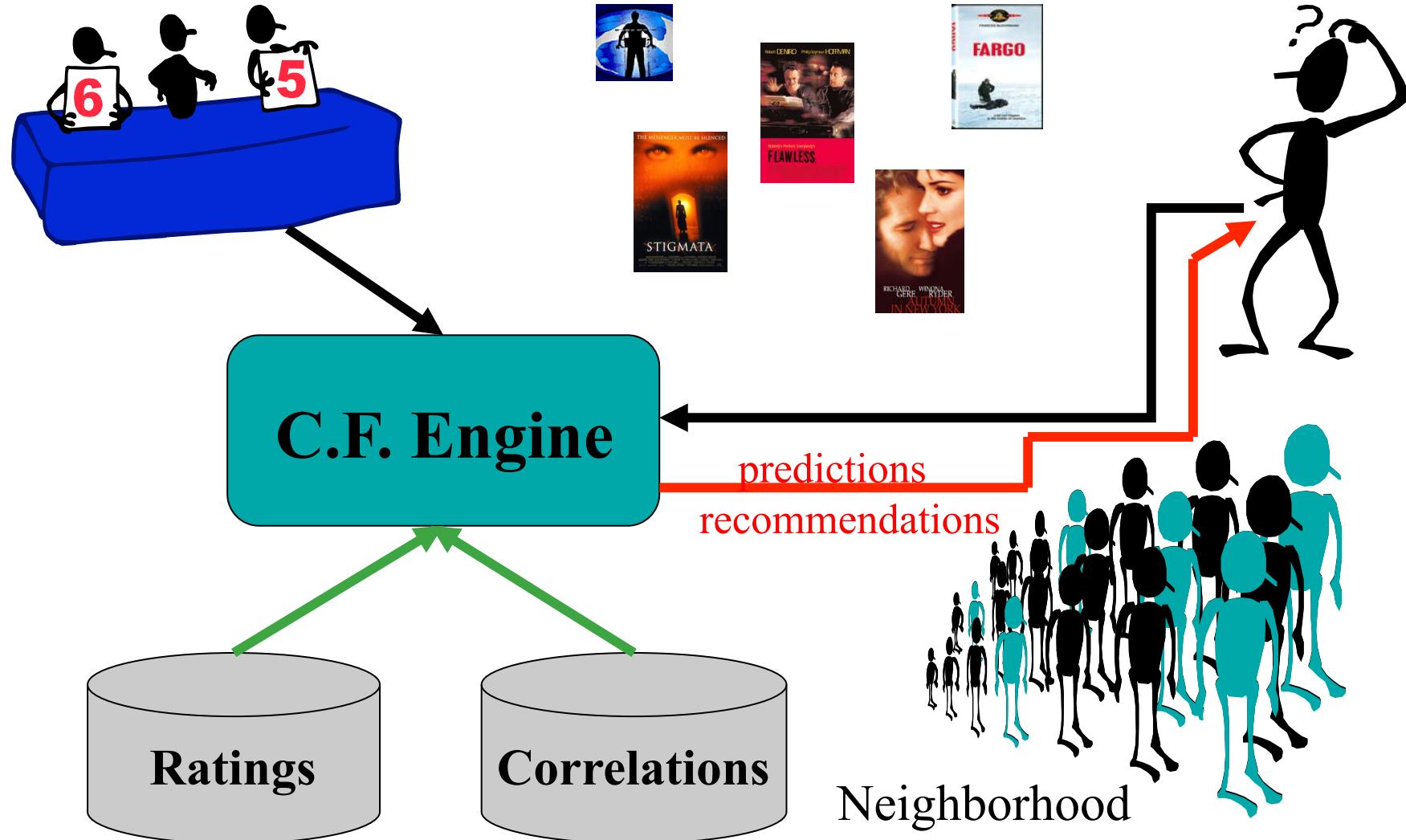
# CF Classic: Request Recommendations



# CF Classic: Identify Neighbors



# CF Classic: Select Items; Predict Ratings



# Understanding the Computation

|        | Hoop Dreams | Star Wars | Pretty Woman | Titanic | Blimp | Rocky XV |
|--------|-------------|-----------|--------------|---------|-------|----------|
| Joe    | D           | A         | B            | D       | ?     | ?        |
| John   | A           | F         | D            |         | F     |          |
| Susan  | A           | A         | A            | A       | A     | A        |
| Pat    | D           | A         |              | C       |       |          |
| Jean   | A           | C         | A            | C       |       | A        |
| Ben    | F           | A         |              |         |       | F        |
| Nathan | D           |           | A            |         | A     |          |

# Understanding the Computation

|        | Hoop Dreams | Star Wars | Pretty Woman | Titanic | Blimp | Rocky XV |
|--------|-------------|-----------|--------------|---------|-------|----------|
| Joe    | D           | A         | B            | D       | ?     | ?        |
| John   | A           | F         | D            |         | F     |          |
| Susan  | A           | A         | A            | A       | A     | A        |
| Pat    | D           | A         |              | C       |       |          |
| Jean   | A           | C         | A            | C       |       | A        |
| Ben    | F           | A         |              |         |       | F        |
| Nathan | D           |           | A            |         | A     |          |

# Understanding the Computation

|        | Hoop Dreams | Star Wars | Pretty Woman | Titanic | Blimp | Rocky XV |
|--------|-------------|-----------|--------------|---------|-------|----------|
| Joe    | D           | A         | B            | D       | ?     | ?        |
| John   | A           | F         | D            |         | F     |          |
| Susan  | A           | A         | A            | A       | A     | A        |
| Pat    | D           | A         |              | C       |       |          |
| Jean   | A           | C         | A            | C       |       | A        |
| Ben    | F           | A         |              |         |       | F        |
| Nathan | D           |           | A            |         | A     |          |

# Understanding the Computation

|        | Hoop Dreams | Star Wars | Pretty Woman | Titanic | Blimp | Rocky XV |
|--------|-------------|-----------|--------------|---------|-------|----------|
| Joe    | D           | A         | B            | D       | ?     | ?        |
| John   | A           | F         | D            |         | F     |          |
| Susan  | A           | A         | A            | A       | A     | A        |
| Pat    | D           | A         |              | C       |       |          |
| Jean   | A           | C         | A            | C       |       | A        |
| Ben    | F           | A         |              |         |       | F        |
| Nathan | D           |           | A            |         | A     |          |

# Understanding the Computation

|        | Hoop Dreams | Star Wars | Pretty Woman | Titanic | Blimp | Rocky XV |
|--------|-------------|-----------|--------------|---------|-------|----------|
| Joe    | D           | A         | B            | D       | ?     | ?        |
| John   | A           | F         | D            |         | F     |          |
| Susan  | A           | A         | A            | A       | A     | A        |
| Pat    | D           | A         |              | C       |       |          |
| Jean   | A           | C         | A            | C       |       | A        |
| Ben    | F           | A         |              |         |       | F        |
| Nathan | D           |           | A            |         | A     |          |

# Understanding the Computation

|        | Hoop Dreams | Star Wars | Pretty Woman | Titanic | Blimp | Rocky XV |
|--------|-------------|-----------|--------------|---------|-------|----------|
| Joe    | D           | A         | B            | D       | ?     | ?        |
| John   | A           | F         | D            |         | F     |          |
| Susan  | A           | A         | A            | A       | A     | A        |
| Pat    | D           | A         |              | C       |       |          |
| Jean   | A           | C         | A            | C       |       | A        |
| Ben    | F           | A         |              |         |       | F        |
| Nathan | D           |           | A            |         | A     |          |

# Understanding the Computation

|        | Hoop Dreams | Star Wars | Pretty Woman | Titanic | Blimp | Rocky XV |
|--------|-------------|-----------|--------------|---------|-------|----------|
| Joe    | D           | A         | B            | D       | ?     | ?        |
| John   | A           | F         | D            |         | F     |          |
| Susan  | A           | A         | A            | A       | A     | A        |
| Pat    | D           | A         |              | C       |       |          |
| Jean   | A           | C         | A            | C       |       | A        |
| Ben    | F           | A         |              |         |       | F        |
| Nathan | D           |           | A            |         | A     |          |

**m o v i e l e n s** - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://movielens.umn.edu/main

Mozilla Firebird Help User Support Forum Plug-in FAQ Kayak LAKAWA- Lakes Area... Yahoo! Calendar - jtr... Slashdot: News for n...

**m o v i e l e n s**  
helping you find the *right* movies

Welcome riedl@cs.umn.edu  
You've rated 205 movies.  
You're the 31st visitor in the past hour.

★★★★★ = Must See  
★★★★☆ = Will Enjoy  
★★★☆☆ = It's OK  
★★☆☆☆ = Fairly Bad  
★☆☆☆☆ = Awful

[Home](#) | [Manage Buddies](#) | [Your Preferences](#) | [Help](#) | [Publish](#) | [Logout](#)

**Shortcuts** **Search**

- [Your Ratings](#)
- [Your Wishlist](#)
- [Newest Additions](#)
- [Rate Random Movies](#)
- [Most Often Rated](#)
- [Suggest Title](#)
- [New Drama](#)
- [New DVDs](#)
- [New Movies](#)

[How to create your own shortcuts](#)

## Welcome to MovieLens!

**Advanced Search** now allows you to search for movies by director and/or actors in addition to its other features: Multiple genres, exclude genres, date ranges, language, hide predictions, and more! Check it out.

Also, don't forget about the new **publish** feature that lets you publish predictions in HTML/RSS 2.0 format. This and other info about recently added features is available in our [archived announcements](#).

**Did you know...?** 4909 people joined MovieLens the same day you did.

| New movies                                     | New DVDs   |
|--|--|
| ★★★★★ Spider-Man 2 (a.k.a. Spiderman 2) (2004) | ★★★★★ Great Escape, The (1963)   |
| ★★★★★ Shrek 2 (2004)                           | ★★★★★ Fog of War: Eleven Lessons from the Life of Robert S. McNamara, The (2003) |
| ★★★★★ Kill Bill: Vol. 2 (2004)                 | ★★★★★ Miracle (2004)   |
| ★★★★★ Anchorman (2004)                         | ★★★★★ Last Samurai, The (2003)   |
| ★★★★★ Super Size Me (2004)                     | ★★★★★ Boat, The (Das Boot) (1981)  |
| ★★★★★ Fahrenheit 9/11 (2004)                   | ★★★★★ Field of Dreams (1989)   |
| ★★★★★ Zatoichi (Zatōichi) (2003)               | ★★★★★ Suddenly (1954)  |
| ★★★★★ Man on Fire (2004)                       | ★★★★★ Lord of the Rings: The Return of the King, The (2003)                      |
| ★★★★★ Mean Girls (2004)                        |  |

Done

**m o v i e l e n s** - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://movielens.umn.edu/main

Mozilla Firebird Help User Support Forum Plug-in FAQ Kayak LAKAWA- Lakes Area... Yahoo! Calendar - jtr... Slashdot: News for n...

**m o v i e l e n s**  
helping you find the *right* movies

Welcome riedl@cs.umn.edu  
You've rated 205 movies.  
You're the 31st visitor in the past hour.

★★★★★ = Must See  
★★★★☆ = Will Enjoy  
★★★★☆ = It's OK  
★★★★☆ = Fairly Bad  
★★★★☆ = Awful

[Home](#) | [Manage Buddies](#) | [Your Preferences](#) | [Help](#) | [Publish](#) | [Logout](#)

**Shortcuts** **Search**

Search Titles  **Go!**  
 Use selected buddies!

Search Genres  
Sci-Fi  2000s   
Domain: All movies   
 Use selected buddies!  
**Search Genres!**

**Advanced Search**

Welcome to MovieLens!

**Advanced Search** now allows you to search for movies by director and/or actors in addition to its other features: Multiple genres, exclude genres, date ranges, language, hide predictions, and more! Check it out.

Also, don't forget about the new **publish** feature that lets you publish predictions in HTML/RSS 2.0 format. This and other info about recently added features is available in our [archived announcements](#).

**Did you know...?** 4909 people joined MovieLens the same day you did.

| New movies  | New DVDs  |
|---|---|
| ★★★★★ Spider-Man 2 (a.k.a.<br>Spiderman 2) (2004) | ★★★★★ Great Escape, The (1963)  |
| ★★★★★ Shrek 2 (2004)                              | ★★★★★ Fog of War: Eleven Lessons from the Life<br>of Robert S. McNamara, The (2003) |
| ★★★★★ Kill Bill: Vol. 2 (2004)                    | ★★★★★ Miracle (2004)  |
| ★★★★★ Anchorman (2004)                            | ★★★★★ Last Samurai, The (2003)  |
| ★★★★★ Super Size Me (2004)                        | ★★★★★ Boat, The (Das Boot) (1981)   |
| ★★★★★ Fahrenheit 9/11 (2004)                      | ★★★★★ Field of Dreams (1989)  |
| ★★★★★ Zatoichi (Zatōichi) (2003)                  | ★★★★★ Suddenly (1954)   |
| ★★★★★ Man on Fire (2004)                          | ★★★★★ Lord of the Rings: The Return of the King,<br>The (2003)                      |
| ★★★★★ Mean Girls (2004)                           |   |

javascript:showTab('Search')

**m o v i e l e n s** - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://movielens.umn.edu/search?searchPhrase=&action=newSearch&hiddenParam=1&genre=

Mozilla Firebird Help User Support Forum Plug-in FAQ Kayak LAKAWA- Lakes Area... Yahoo! Calendar - jtr... Slashdot: News for n...

**m o v i e l e n s**  
helping you find the *right* movies

Welcome riedl@cs.umn.edu  
You've rated 206 movies.  
You're the 31st visitor in the past hour.

★★★★★ = Must See  
★★★★☆ = Will Enjoy  
★★★★☆ = It's OK  
★★★★☆ = Fairly Bad  
★★★★☆ = Awful

[Home](#) | [Manage Buddies](#) | [Your Preferences](#) | [Help](#) | [Publish](#) | [Logout](#)

**Shortcuts** **Search**

Search Titles  **Go!**  
 Use selected buddies!

Search Genres  
Sci-Fi  2000s   
Domain: All movies   
 Use selected buddies!  
**Search Genres!**

Advanced Search

Found 71 movies, sorted by **Prediction**  
Genres: Sci-Fi | Exclude Genres: None  
Dates: 2000s | Domain: All | Format: All | Language: All  
[Show Printer-Friendly Page](#) | [Download Results](#) | [Suggest a Title](#)

Page 1 of 5 [page 2 >](#)

| Predictions for you ↴ | Your Ratings                    | Movie Information   | Wish List                           |
|-----------------------|---------------------------------|---|-------------------------------------|
| ★★★★★                 | Not seen <input type="button"/> | Eternal Sunshine of the Spotless Mind (2004) <a href="#">info</a>   <a href="#">imdb</a><br>Comedy, Drama, Romance, Sci-Fi                                    | <input type="checkbox"/>            |
| ★★★★★                 | Not seen <input type="button"/> | X-Men 2 (a.k.a. X2: X-Men United) (2003) <a href="#">DVD</a> , <a href="#">VHS</a> , <a href="#">info</a>   <a href="#">imdb</a><br>Action, Adventure, Sci-Fi | <input checked="" type="checkbox"/> |
| ★★★★★                 | Not seen <input type="button"/> | Donnie Darko (2001) <a href="#">DVD</a> , <a href="#">VHS</a> , <a href="#">info</a>   <a href="#">imdb</a><br>Drama, Mystery, Romance, Sci-Fi                | <input type="checkbox"/>            |
| ★★★★★                 | Not seen <input type="button"/> | Dune (miniseries) (2000) <a href="#">DVD</a> , <a href="#">VHS</a> , <a href="#">info</a>   <a href="#">imdb</a><br>Drama, Fantasy, Sci-Fi                    | <input type="checkbox"/>            |
| ★★★★★                 | Not seen <input type="button"/> | Spider-Man (a.k.a. Spiderman)   | <input type="checkbox"/>            |

Done

**m o v i e l e n s** - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://movielens.umn.edu/search?searchPhrase=&action=newSearch&hiddenParam=1&genre=

Mozilla Firebird Help User Support Forum Plug-in FAQ Kayak LAKAWA- Lakes Area... Yahoo! Calendar - jtr... Slashdot: News for n...

**m o v i e l e n s**  
helping you find the *right* movies

Welcome riedl@cs.umn.edu  
You've rated 206 movies.  
You're the 31st visitor in the past hour.

★★★★★ = Must See  
★★★★☆ = Will Enjoy  
★★★★☆ = It's OK  
★★★★☆ = Fairly Bad  
★★★★☆ = Awful

[Home](#) | [Manage Buddies](#) | [Your Preferences](#) | [Help](#) | [Publish](#) | [Logout](#)

**Shortcuts**

Search Titles  **Go!**  
 Use selected buddies!

Search Genres  
Sci-Fi  2000s   
Domain: All movies   
 Use selected buddies!  
**Search Genres!**

Advanced Search

Done

Found 71 movies, sorted by **Prediction**  
Genres: Sci-Fi | Exclude Genres: None  
Dates: 2000s | Domain: All | Format: All | Language: All

[Show Printer-Friendly Page](#) | [Download Results](#) | [Suggest a Title](#)

Page 1 of 5 [page 2>](#)

| Predictions for you ↴ | Your Ratings  | Movie Information   | Wish List                           |
|-----------------------|---|---|-------------------------------------|
| ★★★★★                 | Not seen <input type="button"/>   | Eternal Sunshine of the Spotless Mind (2004) <a href="#">info</a>   <a href="#">imdb</a><br>Comedy, Drama, Romance, Sci-Fi                                    | <input type="checkbox"/>            |
| ★★★★★                 | Not seen <input type="button"/><br>Not seen<br>Hide this  | X-Men 2 (a.k.a. X2: X-Men United) (2003) <a href="#">DVD</a> , <a href="#">VHS</a> , <a href="#">info</a>   <a href="#">imdb</a><br>Action, Adventure, Sci-Fi | <input checked="" type="checkbox"/> |
| ★★★★★                 | 0.5 stars<br>1.0 stars<br>1.5 stars<br>2.0 stars<br>2.5 stars<br>3.0 stars <input type="button"/><br>3.5 stars<br>4.0 stars<br>4.5 stars<br>5.0 stars | Donnie Darko (2001) <a href="#">DVD</a> , <a href="#">VHS</a> , <a href="#">info</a>   <a href="#">imdb</a><br>Drama, Mystery, Romance, Sci-Fi                | <input type="checkbox"/>            |
| ★★★★★                 | 3.0 stars <input type="button"/><br>3.5 stars<br>4.0 stars<br>4.5 stars<br>5.0 stars  | Dune (miniseries) (2000) <a href="#">DVD</a> , <a href="#">VHS</a> , <a href="#">info</a>   <a href="#">imdb</a><br>Drama, Fantasy, Sci-Fi                    | <input type="checkbox"/>            |
| ★★★★★                 | 3.0 stars <input type="button"/><br>3.5 stars<br>4.0 stars<br>4.5 stars<br>5.0 stars  | Spider-Man (a.k.a. Spiderman)   | <input type="checkbox"/>            |

**m o v i e l e n s** - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://movielens.umn.edu/search?searchPhrase=matrix&titleSearchBtn=Go%21&action=newSe

Mozilla Firebird Help User Support Forum Plug-in FAQ Kayak LAKAWA- Lakes Area... Yahoo! Calendar - jtr... Slashdot: News for n...

**m o v i e l e n s**  
helping you find the *right* movies

Welcome riedl@cs.umn.edu  
You've rated 206 movies.  
You're the 31st visitor in the past hour.

★★★★★ = Must See  
★★★★☆ = Will Enjoy  
★★★★☆ = It's OK  
★★★★☆ = Fairly Bad  
★★★★☆ = Awful

[Home](#) | [Manage Buddies](#) | [Your Preferences](#) | [Help](#) | [Publish](#) | [Logout](#)

**Shortcuts** **Search**

Search Titles  
 **Go!**  
 Use selected buddies!

Search Genres  
All Genres  All Dates   
Domain: All movies   
 Use selected buddies!  
**Search Genres!**

Advanced Search

You've searched for titles matching: **matrix**  
[Show Printer-Friendly Page](#) | [Download Results](#) | [Suggest a Title](#)

Page 1 of 1

| Predictions for you | Your Ratings                     | Movie Information   | Wish List                |
|---------------------|----------------------------------|---|--------------------------|
| ★★★★★               | 4.5 stars <input type="button"/> | Matrix Reloaded, The (2003) <small>DVD, VHS, info imdb</small><br>Action, Sci-Fi, Thriller    | <input type="checkbox"/> |
| ★★★★★               | 4.0 stars <input type="button"/> | Matrix Revolutions, The (2003) <small>DVD, VHS, info imdb</small><br>Action, Sci-Fi, Thriller | <input type="checkbox"/> |
| ★★★★★               | 5.0 stars <input type="button"/> | Matrix, The (1999) <small>DVD, VHS, info imdb</small><br>Action, Sci-Fi, Thriller             | <input type="checkbox"/> |

Page 1 of 1

Create a shortcut to this search!  
Enter a name then press "Create!"  **Create!**  
[What are shortcuts?](#)

Done

**movielens - Mozilla Firefox**

File Edit View Go Bookmarks Tools Help

http://movielens.umn.edu/search?genreSearch=1&genre=All&format=All&exGenre=None&date

Mozilla Firebird Help User Support Forum Plug-in FAQ Kayak LAKAWA- Lakes Area... Yahoo! Calendar - jtr... Slashdot: News for n...

**Shortcuts** **Search**

Search Titles  **Go!**  
 Use selected buddies!

Search Genres  
All Genres  All Dates   
Domain: DVD releases   
 Use selected buddies!  
**Search Genres!**

Advanced Search

Select Buddies  
 Joe  
 Maureen  
**What are buddies?**

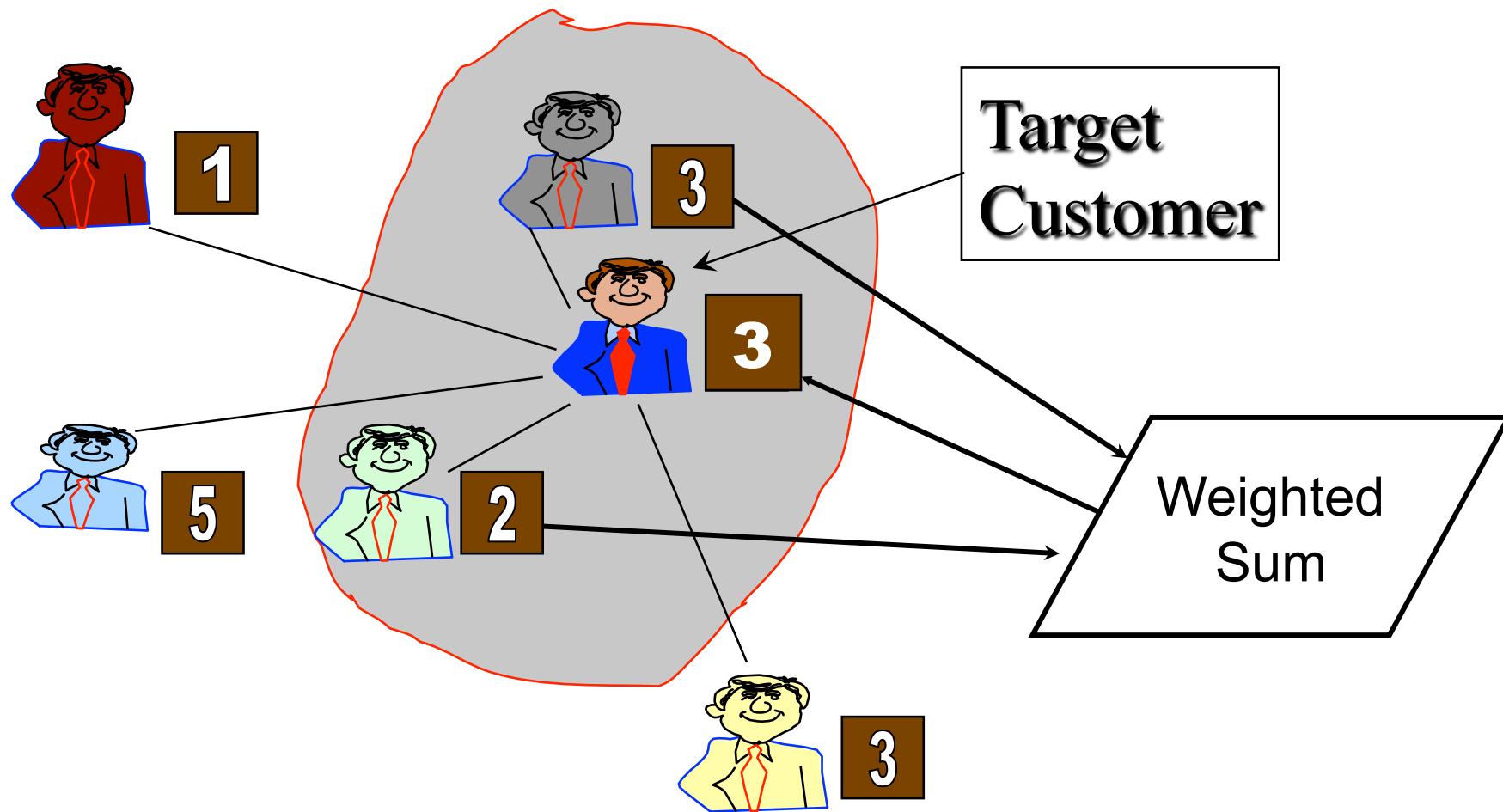
Found 4511 movies, sorted by **Prediction**  
Genres: All | Exclude Genres: None  
Dates: All | Domain: DVD Releases | Format: All | Language: All  
[Show Printer-Friendly Page](#) | [Download Results](#) | [Suggest a Title](#)

Buddy Search with: **Joe, Maureen**  
Results restricted to movies that have predictions for each buddy.

Page 1 of 301 | Go to page:  
1...60...120...180...240...300...last [page 2>](#)

| Combined<br>Preds | You | Joe | Maureen | Your<br>Ratings                 | Movie<br>Information  | Wish<br>List             |
|-------------------|-----|-----|---------|---------------------------------|---|--------------------------|
| ★★★★★             | 4.5 | 4.5 | 4.5     | Not seen <input type="button"/> | <b>Scrooge (1951)</b><br>DVD, VHS, info   imdb<br>Drama, Fantasy                    | <input type="checkbox"/> |
| ★★★★★             | 4.5 | 4.5 | 4.5     | Not seen <input type="button"/> | <b>Decade Under the Influence, A (2003)</b><br>DVD, VHS, info   imdb<br>Documentary | <input type="checkbox"/> |
| ★★★★★             | 4.5 | 4.5 | 4.5     | Not seen <input type="button"/> | <b>Miracle Worker, The (1962)</b><br>DVD, VHS, info   imdb<br>Drama                 | <input type="checkbox"/> |
| ★★★★★             | 4.5 | 4.5 | 4.5     | Not seen <input type="button"/> | <b>Enter the Dragon (1973)</b><br>DVD, VHS, info   imdb<br>Action, Crime, Drama     | <input type="checkbox"/> |

# User-User Collaborative Filtering



# Recommenders

- Tools to help identify worthwhile stuff
  - Filtering interfaces
    - E-mail filters, clipping services
  - Recommendation interfaces
    - Suggestion lists, “top-n,” offers and promotions
  - Prediction interfaces
    - Evaluate candidates, predicted ratings

# A Little Vocabulary

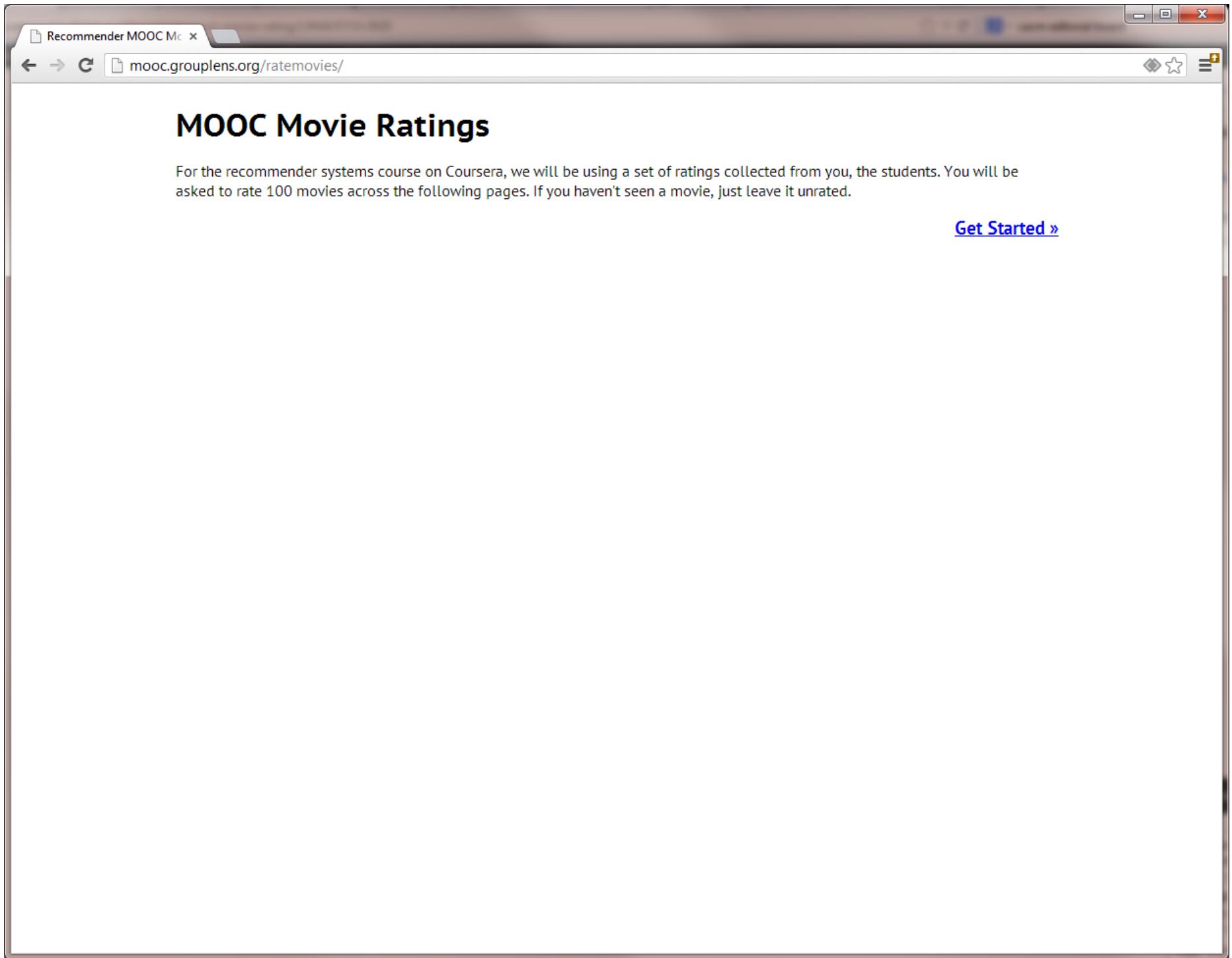
- Rating – expression of preference
  - Explicit rating (direct from the user)
  - Implicit rating (inferred from user activity)
- Prediction – estimate of preference
- Recommendation – selected items for user
- Content – attributes, text, etc.
- Collaborative – using data from other users

# Historical Challenges

- Collecting Opinion and Experience Data
- Finding the Relevant Data for a Purpose
- Computing the Recommendations
- Presenting the Data in a Useful Way

# Your First Assignment

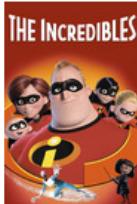
- We are building a class ratings dataset using the MovieLens infrastructure
  - This will be used for several of the assignments
- Your assignment is to rate movies through our interface:
  - <http://mooc.grouplens.org/ratemovies/>



Recommender MOOC Mooc

mooc.groupLens.org/ratemovies/page/1

## Movie Page 1

 **The Incredibles (2004)**  
Pixar Animation Studios and Walt Disney Pictures  
Directed by Brad Bird  
*No gut, no glory.*  

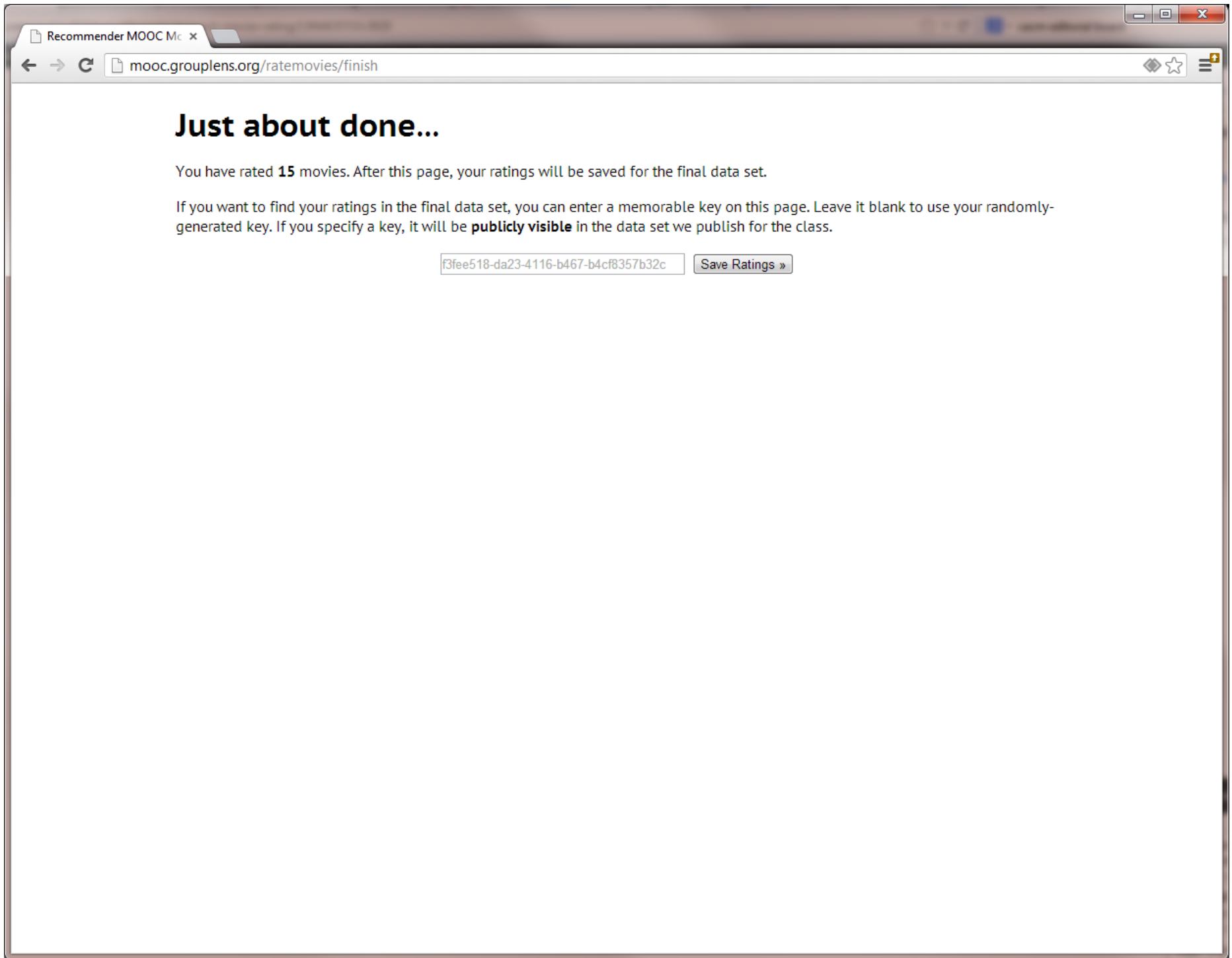

 **Gone In Sixty Seconds (2000)**  
Jerry Bruckheimer Films  
Directed by Dominic Sena  
*Ice Cold, Hot Wired.*  


 **Lost in Translation (2003)**  
American Zoetrope  
Directed by Sofia Coppola  
*Everyone wants to be found.*  


 **Iron Man (2008)**  
Paramount Pictures, Marvel Entertainment and Marvel Studios  
Directed by Jon Favreau  
*Heroes aren't born. They're built.*  


 **28 Days Later... (2002)**  
British Film Council, Fox Searchlight Pictures and DNA Films  
Directed by Danny Boyle  
*His fear began when he woke up alone. His terror began when he realised he wasn't.*  


 **My Big Fat Greek Wedding (2002)**



# Your First Assignment

- We are building a class ratings dataset using the MovieLens infrastructure
  - This will be used for several of the assignments
- Your assignment is to rate movies through our interface:
  - <http://mooc.grouplens.org/ratemovies/>
  - Only rate movies you've seen
  - You can create a key to follow your preferences forward (or save the random key)

# Moving Forward ...

- Next Lecture: Formal Course Introduction
- Rest of this Week
  - Technical details for programmers
  - Taxonomy
  - Tour of Recommenders in Amazon.com
- If not familiar with recommenders at all:
  - Spend some time using them ([movielens.org](http://movielens.org))

# 1-1: Introduction to Recommender Systems

Introduction to Recommender Systems

# 1-2: Welcome to the Course!

# Goals for Today

- To understand the scope of the course.
- To understand the two tracks and the prerequisites for each.
- To understand the dual MOOC/U of M course structure and its implications
- To review and understand basic course concepts including assignments, exams, expectations, grading, academic standards, communication and feedback, and more ...

# Introductions

- Joseph A. Konstan
- Michael Ekstrand

# Dedication

- John Riedl
  - Pioneer in Recommender Systems



# Course Scope

- Broad Overview of Recommendation Techniques
  - Focused around algorithms
    - Non-personalized, content-based, collaborative
    - Emphasis on personalized, collaborative
  - Added topics/enrichment around interaction and design or recommender systems
  - Goal is to provide solid algorithmic core with extensive awareness of related topics
  - We include interviews with many leaders in field

# Programming Challenge I

- Understanding recommender systems is useful even to people who can't program them.
- Approach: two tracks
  - Concepts track: expected to complete non-programming assignments, exams
  - Programming track: expected to have significant Java programming skill

# Recommended Background ...

- All Students
  - College-level algebra
  - Basic computing concepts and skills
- Programming Track
  - Java programming (extensive)
  - Data structures
  - Ability to install/manage open source software tools and libraries

# Programming Challenge II

- Building quality recommender algorithms is hard – and often the details are best re-used rather than re-created
- Focus on using LensKit toolkit (and extending/adapting where possible) as a way to explore more algorithms with less programming effort
  - Big benefit: can then use LensKit afterwards (open source Recommender toolkit)

# Mix of Coursera and Credit Students

- This course is being taken simultaneously by U of M undergrad/graduate students
  - U of M students all must be in the programming track
  - U of M students will participate in all online activities (including online grading)
- U of M students will have regular face-to-face sessions
  - *Question-driven sessions will be recorded and posted for Coursera students to view (optional)*

# Interaction ... the Class Forums

- There are a lot of you (almost 8000 one month before launch!)
- We will not be taking questions directly – all questions must come via the class forums
  - Organized by topic (technical, programming, course modules, assignments)
  - Be sure to vote up questions you feel most deserve answers
  - We will post replies to top vote-getters at least twice a week

# Course Requirements and Grading

- Grades based on assignments and exams
- Two exams (multiple choice/short answer)
- “Written” and “Programming” assignments
  - Six pairs of assignments (one per two-week module), plus “assignment zero” this week
  - Variety of grading techniques (automatic, peer)
  - Get an early start (final deadlines are real)

# Assignment grading

- We are grading results, not process
- You are generally asked to submit a file (usually csv format) and we'll provide a sample or specifications
- Some programming assignments come with code to submit the file themselves
- For many assignments, you'll get a personal set of test cases to submit

# Written Assignment Zero ...

- Use the following website to rate a set of movies ... this will build a class dataset for use in our later assignments.  
<http://mooc.grouplens.org/ratemovies/>
- When you're done, you'll receive a code to enter into Coursera – it's that easy!
- You can use the code (or add your own identifier) to follow your profile through the course.

# Statements of Accomplishment

- Standard: 50% of possible points (approx. 80% of non-programming points)
- Distinguished: 80% of possible points (requires completing programming assts.)

Note: U of M registered students will get regular grades

# Academic Standards

- Academic integrity is essential
- Honor code online
  - U of M academic standards for credit students
- All written assignments and exams must be your own work
- Programming assignments may be completed in pairs, but each student must generate own output file; collaborators must be identified as part of online submission

# Feedback and Surveys

- We're all still learning, and we will be studying how this course goes both to make mid-course corrections (where possible) and to shape future offerings
- Please participate in surveys and provide feedback
  - We have partnered with education researchers to analyze data and learn from the experience
- We will also have a “general suggestions” topic in the class forum; vote up good ideas!

# Final Thoughts

- We're glad you're here
- Own your educational experience
  - No stigma associated with view-only
- But commit enough time to get value
  - Programming track will take 10+ hours a week for many students

# 1-2: Welcome to the Course!

# Software Environment

# Programming Asn. Requirements

- Java (JDK 7 recommended, 6 works)
- Apache Maven
- LensKit
- An IDE or editor
- Additional software may be useful
  - R
  - GraphViz

# This video: Windows setup

- Set up Java, Maven, LensKit, IntelliJ
- On Windows
  - It's the hardest
  - Linux and Mac instructions online
- Next video: Eclipse setup
- Written instructions online

# Software Environment

# 1-4(a): Taxonomy of Recommender Systems (part 1 of 2)

# Learning Objectives

- To understand the different types of recommender systems
  - A framework for analyzing recommender systems in general
  - A specific overview of different recommendation algorithms
- To acquire a roadmap for the rest of the course, based on the algorithms studied

# Analytical Framework

- Dimensions of Analysis
  - Domain
  - Purpose
  - Recommendation Context
  - Whose Opinions
  - Personalization Level
  - Privacy and Trustworthiness
  - Interfaces
  - Recommendation Algorithms

# Domains of Recommendation

- Content to Commerce and Beyond
  - News, information, “text”
  - Products, vendors, bundles
  - Matchmaking (other people)
  - Sequences (e.g., music playlists)
- One particularly interesting property
  - New items (e.g., movies, books, ...)
  - Re-recommend old ones (e.g., groceries, music)



pagerank



SIGN IN

[Web](#)[Images](#)[Maps](#)[Shopping](#)[Applications](#)[More](#) ▾[Search tools](#)

About 247,000,000 results (0.31 seconds)

Ad related to pagerank ⓘ

[Moz Official Site - moz.com](#)[www.moz.com/pro](#)

Check ranks, understand links, and help your sites traffic. Free trial

Moz has 33,862 followers on Google+

Take a 30 day Free Trial - Learn SEO

[PageRank - Wikipedia, the free encyclopedia](#)[en.wikipedia.org/wiki/PageRank](#) ▾

PageRank is a link analysis algorithm, named after Larry Page and used by the Google web search engine, that assigns a numerical weighting to each element ...

[HITS algorithm - Google Toolbar - Rajeev Motwani - Webgraph](#)[Google PageRank Checker - Check Google page rank instantly](#)[www.prchecker.info/check\\_page\\_rank.php](#) ▾

Page Rank Checker is a completely free service to check Google pagerank instantly using our online page rank check tool or a small pagerank button.

[Check Page Rank!](#)[www.checkpagerank.net](#) ▾

NET - FREE TOOL TO CHECK GOOGLE PAGE RANK ! Google PageRank (Google PR) is one of the methods Google uses to determine a page's relevance or ...

[Check PageRank - SEO - Google Penalty Recovery ... - Add Our Free Page Rank ...](#)[What Is PageRank - PageRank Explained - Google - About.com](#)[google.about.com](#) > ... > Search Engine Optimization

by Marziah Karch - in 6,474 Google+ circles

PageRank is what Google uses to determine the importance of a web page. It's one of many factors used to determine which pages appear in search results.

[PageRank - Toolbar Help](#)<https://support.google.com/toolbar/answer/79837?hl=en>

Mar 29, 2013 – Pause your cursor over the PageRank button to display the importance of the webpage you're viewing according to Google. Webpages with a ...

## Introduction to Recommender Systems

[Chrome Web Store - PageRank Status](#)<https://chrome.google.com/webstore/detail/pagerank/bhdkkfboekddppnqjgohbmnbh>

# Hammacher Schlemmer

Offering the Best, the Only, and the Unexpected for 165 years

Customer Service 800-321-1484 | Order Tracking | Catalog Quick Order



0 item(s)

New Arrivals • Our Favorites • Gift Guide • The Best • The Only • The Unexpected • Request Catalog

 search keyword or item #

Search

[Electronics](#) [Apparel](#) [Home Living](#) [Outdoor Living](#) [Personal Care](#) [Sports & Leisure](#) [Toys](#) [Travel](#) [Special Values](#)
[Home](#) » [Sports & Leisure](#) » [Games](#)


Hover and click to magnify, click again to zoom.



## The Lexicographer's Extended Scrabble.

This is the version of Scrabble with nearly twice the squares and letters as the standard game to extend play for true lexicon aficionados. With a 21 x 21 grid—441 spaces—this board provides Scrabble enthusiasts more room to unload their letter tiles than the traditional 15 x 15 grid, possibly to modify an opponent's Scrabble-legal QUANT into a devastating QUANTIFY Quadruple Word Score. To facilitate extended game play, this set includes 200 plastic letter tiles, twice as many as a standard game, which also allows cruciverbalophiles to create words not possible with standard Scrabble. The 14 1/2" square game board can be angled and rotated so all players get an equally clear view, yet has pegs around each letter space that hold tiles firmly in place. For two to four players. Ages 8 and up. 2" H x 14 1/2" W x 14 1/2" L. (3 lbs.)

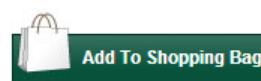
[Description](#)[Reviews](#) [Lifetime Guarantee](#)

Item 81810

Price \$39.95

How Many? 

Available for Immediate Shipment.

Add Gift Wrap for \$6.95 

May we recommend:



The Giant Monopoly Game.



The Advanced Electronic Crossword Puzzle Dictionary.



The Touchscreen Poker Game.



The World's Largest Scrabble Game.

# Purposes of Recommendation

- The recommendations themselves
  - Sales
  - Information
- Education of user/customer
- Build a community of users/customers around products or content

# OWL Tips

| OWL Tips     |   |   |                                   |
|--------------|---|---|-----------------------------------|
| Date 9/30/98 | Word Commands                           | Comments  | Priority                          |
|              | EditFind                                | Use Find more to search for text in file        | 88                                |
|              | EditDeleteWord                          | Learn the shortcut keys to delete words         | 87                                |
|              | FormatUnderline                         | Try using underlining for formatting text       | 80                                |
|              | FileClose                               | Try different ways to close your file           | 74                                |
|              | EditReplace                             | Use Replace more for finding and replacing text | 72                                |
|              | ViewZoom                                | Learn how to enlarge or reduce the display      | 69                                |
|              | ViewShowAll                             | FYI-more than average use for Show All commar   | 67                                |
|              | FormatBulletsAndNumbering               | Learn how to automatically add bullets and numt | 64                                |
|              | ToolsWordCount                          | Use Word Count to look up statistics on files   | 61                                |
|              | ViewPage                                | Use Page Layout to view files before printing   | 57                                |
| User M06375  | OWL Version 5.0c                        |   |                                   |
|              | <input type="button" value="OWL Help"/> |   | <input type="button" value="OK"/> |



Eleanor Steinman and Ramesh Jain are friends with reviewers of Mandarin Oriental, Las Vegas

## Mandarin Oriental, Las Vegas ★★★★☆

Special Offer 3rd Night Free



Ranked #1 of 271 hotels in Las Vegas

5 reviews

"a truly deserve 1st place !!" 06/27/2013

"oasis of tranquility" 06/27/2013

[Professional photos](#) | [Traveler photos \(951\)](#) | [Map](#)

[Price Your Stay](#)



5 reviews  
30% off room rate



Comfort Inn Airport  
5 reviews  
Free Breakfast and Wi-Fi

[View all Special Offers in Las Vegas](#)



Elizabeth Churchill and Dan Bodenheimer are friends with reviewers of Four Seasons Hotel Las Vegas

## Four Seasons Hotel Las Vegas ★★★★☆

Special Offer USD 100 Hotel Credit



Ranked #2 of 271 hotels in Las Vegas

5 reviews

"Superior Hotel on the Vegas Strip" 06/28/2013

"Luxury in Sin City" 06/28/2013

[Professional photos](#) | [Traveler photos \(648\)](#) | [Map](#)

[Price Your Stay](#)



Bellagio Las Vegas  
5 reviews  
Las Vegas, NV



Venetian Resort Hotel Casino  
5 reviews  
Las Vegas, NV



Mandarin Oriental, Las Vegas  
5 reviews  
Las Vegas, NV

## Staybridge Suites Las Vegas ★★★★☆

Special Offer Save on Hotel Packages!



Ranked #3 of 271 hotels in Las Vegas

5 reviews

"Best hotel!" 06/26/2013

"Very Nice Hotel off the strip" 06/25/2013

[Professional photos](#) | [Traveler photos \(225\)](#) | [Map](#)

[Price Your Stay](#)



The Strip  
Andrew Fetterer rated this attraction  
5 reviews



Springs Preserve  
J B Lawton III likes this attraction  
5 reviews



Bellagio Fountains  
Andrew Fetterer rated this attraction  
5 reviews

## ARIA Sky Suites



Ranked #4 of 271 hotels in Las Vegas

5 reviews

"A unique thing of beauty." 06/28/2013

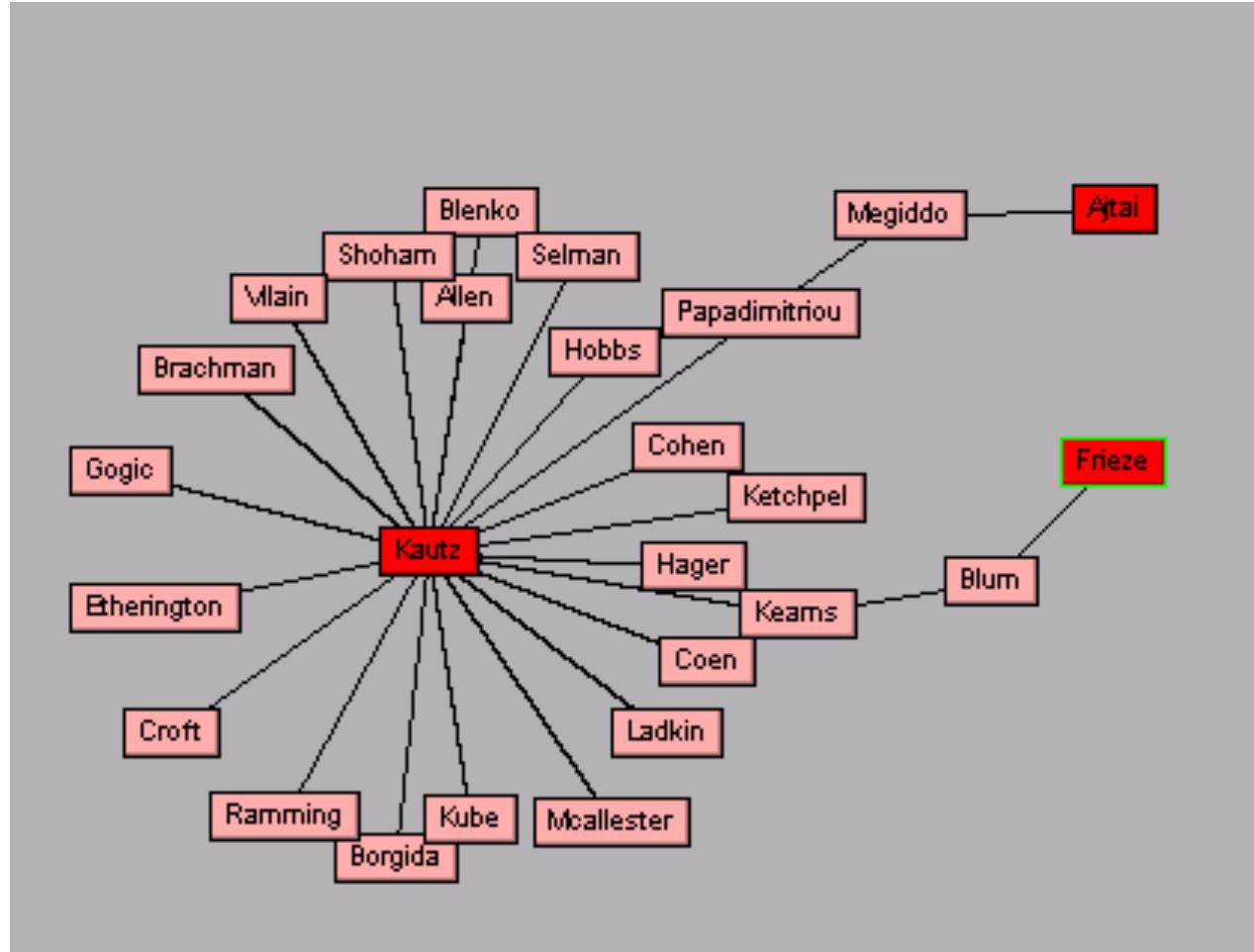
"Serengeti retreat above the Vegas Strip" 06/27/2013

[Traveler photos \(121\)](#) | [Map](#)

37 friends have been to Las Vegas & nearby towns



# ReferralWeb



# Recommendation Context

- What is the User doing at the time of recommendation?
  - Shopping
  - Listening to Music
  - Hanging out with other people
- How does the context constrain the recommender?
  - Groups, automatic consumption (vs. suggestion), level of attention, level of interruption?

PANDORA®

Billy Joel Radio

New Station

Type in artist, genre, or composer



0:18

-2:55

Goodbye Yellow Brick Road  
by Elton John  
on Goodbye Yellow Brick Road[Now Playing](#)[Music Feed](#)[My Profile](#)[Shuffle](#)[Billy Joel Radio](#)[add variety](#)[options ▾](#)

### Billy Joel Radio

From here on out we'll be exploring other songs and artists that have musical qualities similar to Billy Joel. This track, "Goodbye Yellow Brick Road" by Elton John, has similar mellow rock instrumentation, a subtle use of vocal harmony, acoustic rhythm piano, intricate melodic phrasing and mixed acoustic and electric instrumentation.

**Goodbye Yellow Brick Road**  
by Elton John  
on Goodbye Yellow Brick Road

[Publish ▾](#) [Share...](#) [Buy ▾](#)

### Lyrics

When are you gonna come down  
When are you going to land  
I should have stayed on the farm  
I should have listened to my old man

[show more ▾](#)

### About Elton John

In terms of sales and lasting popularity, Elton John was the biggest pop superstar of the early '70s. Initially marketed as a singer/songwriter, John soon revealed he could craft Beatlesque pop and pound out rockers with

[full bio](#)

### Similar Artists

[The Beatles](#)[Eagles](#)

## Introduction to Recommender Systems

[Pandora for your](#)

Your Ad Here

### XFINITY® TRIPLE PLAY EXCLUSIVE ONLINE OFFER

GET STARTED AT  
**\$99**  
A MONTH FOR 12 MONTHS

\$100 VISA®  
PREPAID CARD

[LEARN MORE ▾](#)

Ads by Google

### 2013 MAZDA3 SKYACTIV

Up to 40 MPG Highway.  
Efficiency & Performance all in one  
[www.MazdaUSA.com](#)

### Nursing School Online

Earn Your RN To BSN in 3 Semesters.  
Online, Affordable & User Friendly!  
[www.Chamberlain.edu](#)

Want to go back on  
**BIRTH CONTROL?**

[why ads?](#)[skip](#)

# Whose Opinion?

- “Experts”
- Ordinary “phoaks”
- People like you



My Account ? Help Shopping Cart

Home

Wine Shop

Rare Wines

Wine Clubs

Gift Shop

Corporate Gifts

Accessory Shop

Message Boards

Search:

Go

## Wine Selector:

Category

Price

Origin

Go

## More Search Options ▶

## Short Cuts:

Red Wines

CabernetMerlotZinfandel

White Wines

Chardonnay

Bubbly Wines

Rare Wines

! What's New

Samplers

\* Specials

P Peter's Picks

※ Bang for the Buck

Personalized WineWineriesWine Team

## Live Help

Have a question for  
Customer Service? Use Live  
Help to get it answered  
without going online.Introduction to Recommender Systems  
Monday-Sunday: 8 a.m. to 5 p.m.

## Casa Lapostolle

1997 Casa Lapostolle Cabernet  
Sauvignon, Rapel Valley, Chile

Lapostolle's 1997 Cabernet Sauvignon is quite ripe and delicious, showing the soft tannins and easy-drinking profile needed to pair with everything from meatloaf to spaghetti and meatballs.

\$9.95

Add to Cart ▶

[Add to My Wish List](#)

## Peter's Tasting Chart

|                |   |
|----------------|---|
| intensity ▶    | delicate ----- ● ----- powerful         |
| dry or sweet ▶ | bone dry ----- ● ----- dessert          |
| body ▶         | light body ----- ● ----- very full body |
| acidity ▶      | soft, gentle ----- ● ----- very crisp   |
| tannin ▶       | none ----- ● ----- heavy tannins        |
| oak ▶          | none ----- ● ----- heavy oak            |
| complexity ▶   | direct ----- ● ----- very complex       |

Casa Lapostolle, a partnership between French and Chilean winemaking families, is one of the newer wineries in Chile but they clearly know how to make fine wine. A red that rewards regular visits, Casa Lapostolle's 1997 Cabernet Sauvignon is so juicy, easy to drink, and affordable that you may want to buy it by the case. Ripe flavors of cassis and plum fruit are aligned within a delicate frame of oak and a long, silky-tannic finish that's perfect for everyday home-style comfort foods. The tannins are soft enough and the

Our Wine Experts  
Recommend...

1997 Domaine Bascou  
Vin de Pays d'Oc  
(Cabernet Sauvignon),  
France

A thoroughly modern,  
international-style red.  
**\$9.95 ▶**

1997 Meridian Cabernet  
Sauvignon, California

A mouthful of ripe black  
cherry fruit.  
**\$11.00 ▶**

1998 Errazuriz Cabernet  
Sauvignon El Ciebo  
Estate, Aconcagua  
Valley, Chile

A deliciously ripe and  
complex Cabernet  
Sauvignon.  
**\$8.95 ▶**

## Related Wines

Cabernet SauvignonCasa LapostolleChile

# People Helping One Another Know Stuff

Freq "Together, we know it all." [Pheedback](#)

[Recency](#) [Top](#)  
[Posters](#)

[Area](#) [Help](#)  
[Summary](#)

\* Searches for posted web pages that contain any of the above words

Navigate Up: [PHOAKS Home Page](#) : [Newsgroup Areas](#) : [rec . music . dylan](#)

## Frequently Mentioned Resources

| <i>Resource Title</i>  | <i>Distinct Posters</i> | <i>Click on Bars for Message Context(s)</i> * |
|--|-------------------------|---|
| 1) <a href="#">Bob Dylan - Bob Links</a>                         | <a href="#">23</a>      | ...   |
| 2) <a href="#">Bob Dylan Chords</a>                              | <a href="#">9</a>       |   |
| 3) <a href="#">RemarQ - The Internet's Best Collaboration...</a> | <a href="#">8</a>       |   |
| 4) <a href="#">bobdylan.com: Bob Dylan</a>                       | <a href="#">7</a>       |   |
| 5) <a href="#">CDNOW</a>   | <a href="#">6</a>       |   |
| 6) <a href="#">Mailing List WWW Gateway</a>                      | <a href="#">6</a>       |   |
| 7) <a href="#">Deja.com</a>                                      | <a href="#">4</a>       |   |
| 8) <a href="#">LC Z39.50 Server Soft Reference</a>               | <a href="#">4</a>       |   |
| 9) <a href="#">Resource at www.cs.umass.edu</a>                  | <a href="#">4</a>       |   |
| 10) <a href="#">Sidewalk</a>                                     | <a href="#">4</a>       |   |

\* Note: each square represents the posting of one resource (e.g., URL) by one person. The lighter the square, the more recent the post. Click on a square to view messages where this resource was mentioned. Posting a web resource does not necessarily imply endorsing that resource. Sometimes it may actually mean the opposite. Consult the relevant netnews messages to obtain context.

# Personalization Level

- Generic / Non-Personalized
  - Everyone receives same recommendations
- Demographic
  - Matches a target group
- Ephemeral
  - Matches current activity
- Persistent
  - Matches long-term interests



1-800-963-4816

Lands' End Store

Overstocks

Corporate Sales

Search for

go

In All Products

Catalog Quick Order

go

Enter an Item Number

Sign up!

- [Subscribe to our e-mail newsletter](#)
- [Request our Catalog](#)
- [Join our Affiliate program](#)

Special Services

- [Gift Certificates](#)
- [Track your Order status](#)

International Sites

- [Shop in your local language](#)

Our Company

- [General Information](#)
- [Investor's Corner](#)

Introduction to Recommender Systems

Shopping Bag Checkout My Account My Model Personal Shopper Ask Us

Welcome Women's | Men's | Kids' | For the Home | Luggage | Gifts

# It makes looking good look easy.

## *The slimming Faille Tankini – just \$58!*

### AS SEEN ON TV!

The magic word is Faille (say it "file"). It's a revolutionary ribbed fabric that feels slimming and comfortable.

With a liberating – yet discreet – 2-piece style, our [Faille Tankini](#) works its magic at a very down-to-earth price: \$58.

Want a pile of faille? See [all of our slimming Faille favorites](#).



Faille Tankini

### Let Swim Finder locate your perfect suit!

Quickly sorts through hundreds of Suits by:

- Body Shape • Anxiety Zones • Leg Height •
- Bra Style • 18W-26W • Mastectomy • and more!

It's fast...it's fun! [Try Swim Finder today!](#) Or, visit [Swim HQ](#) to see new styles, swim accessories and more.



### Important Notice!

Lands' End has agreed to be acquired by Sears.

Read the [announcement by Lands' End CEO David Dyer](#), and see the companies' [joint press release](#).



Design your own pants right here:

- [New! Men's Custom Jeans](#)
- [Men's Custom Chinos](#)
- [New! Women's Custom Jeans](#)
- [Women's Custom Chinos](#)

### Subscribe!

News! Specials! Enter your e-mail address to subscribe:

go

### New! Lands' End Maternity

Internet Exclusive!  
[View Details](#)

**B**

Brooks Brothers - Classically Modern Men's and Women's Apparel - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Search Favorites History

Address http://www.brooksbrothers.com/ Go

productsearch go shoppingbag

enter an item number or keyword to find your product

Welcome! Please [sign-in](#) or [register](#) with us and save \$10!

men women clearance specialtyshops

*Brooks Brothers*

light, luxurious **linen** **save 25%**  
on all men's and women's linen through 5.27  
[VIEW THE COLLECTION](#)

**THE GOLDEN FLEECE POLO**  
**NOW 3 FOR \$85**  
[COMPARE & SAVE](#)

**BECOME A MEMBER & SAVE**  
faster and easier shopping plus \$10 OFF  
[REGISTER TODAY](#)

**MEN'S & WOMEN'S CLEARANCE**  
**UP TO 80% OFF**  
[BEGIN SAVING](#)

brooks.buys gifts top ten boys brooks.mail about us 1.800.556.7039

emailsignup submit stores catalog brooks card privacy help members | sign-in

https://www.brooksbrothers.com/dynaset.asp?html=members&secure=yes

Internet



**Folk & Blues up to 30% off**  
Save on Country music, too!

**Stay Awake at Work with**  
**Non-Stop Music from CDNOW Radio**

MUSIC

VIDEO

GIFTS

MY CDNOW

HELP

[Gift Certificates](#)[Album Advisor](#)[Box Sets](#)[Movies as Gifts](#)

[Shopping Cart](#)  
contains 0 items

Artist

[Search Classical](#)

[Rock](#)  
[Alternative/Indie](#)  
[Pop/R&B](#)  
[Hip-Hop](#)

[Electronic/Dance](#)[Jazz](#)[Country](#)[Folk/Blues](#)[World](#)[Latin](#)[Classical](#)[New Age](#)[Christian/Gospel](#)[Vocal/Theatrical](#)[Soundtracks](#)[Comedy/Spoken](#)[Kids/Family](#)[MTV CD Lounge](#)[VH1 Music Shop](#)[Album Advisor™](#)[Home](#)

### ❖ **Album Advisor™**

Tell us what you like and we'll make several recommendations. Great for buying gifts or broadening your musical horizons.

To start, enter the names of up to three artists  
below and click on the **Recommend** button. ↴

gordon bok

enya

[Introduction to Recommender Systems](#)

Artist

[Search Classical](#)

[Ace Of Base](#)[Cruel Summer](#)

List \$16.97

[Add to Cart \\$12.99](#) [Listen](#)["Cruel Summer"](#)[Real Audio](#)[Windows Media](#)[See complete track list and more album info.](#)[Fantasia 2000](#)[Score](#)

List \$17.97

[Add to Cart \\$13.99](#) [Listen](#)["Symphony No.5"](#)[Real Audio](#)[See complete track list and more album info.](#)[Erasure](#)[Abba-Esque](#)

List \$6.97

[Add to Cart \\$6.89](#) [Listen](#)["Lay All Your Love On Me"](#)[Real Audio](#)[Windows Media](#)[See complete track list and more album info.](#)[Vangelis](#)[Opera Sauvage](#)[List \\$11.49](#)[Add to Cart \\$11.49](#) [Listen](#)["Hymne"](#)



MUSIC VIDEO GIFTS MY CDNOW HELP

Account

Order History

Wish List

CDNOW Recommends

 Shopping Cart  
contains 0 items

Artist



Search Classical

## Summary for John Riedl



## ORDER HISTORY

Order  
**# 14856068**

Placed on  
December 25,  
1999

Status: Order  
Shipped

[Get additional  
order information.](#)

## CDNOW RECOMMENDS

[John Coltrane](#)[Love Supreme](#)

List Price \$17.97

[Add to Cart \\$13.99](#)

[More items](#) recommended for you!

## WISH LIST

Want to keep track of items you might like to purchase at a later date?

Start [your list](#) today!

## REWARDS

Current Program:  
**None**

You are not currently signed up for any rewards program.  
[Start earning rewards for FREE CDs and more right now!](#)



## FAVORITE ARTISTS

Bok\*Gordon / Muir\*Ann Mayo / Trickett\*Ed

Modify your options and check for Advance Orders and New Releases from [your favorite artists](#).



## ACCOUNT INFORMATION &amp; ADDRESS BOOK

Modify [your Account Information](#), change your [Primary Address](#) or set up [Express Checkout](#).

## PREFERENCES

[Introduction to Recommender Systems](#)  
[Customize your](#)



## RATE YOUR MUSIC

You have 3 items that you haven't rated yet.

Tell us  about the music you own and help us make better recommendations!



# Privacy and Trustworthiness

- Who knows what about me?
  - Personal information revealed
  - Identity
  - Deniability of preferences
- Is the recommendation honest?
  - Biases built-in by operator
    - “business rules”
  - Vulnerability to external manipulation
  - Transparency of “recommenders”; Reputation

# Interfaces

- Types of Output
  - Predictions
  - Recommendations
  - Filtering
  - *Organic vs. explicit presentation*
    - Agent/Discussion Interface
- Types of Input
  - Explicit
  - Implicit

# Recommendation Algorithms

- Non-Personalized Summary Statistics
- Content-Based Filtering
  - Information Filtering
  - Knowledge-Based
- Collaborative Filtering
  - User-User
  - Item-Item
  - Dimensionality Reduction
- Others
  - Critique / Interview Based Recommendations
  - Hybrid Techniques

# 1-4(b): Taxonomy of Recommender Systems (part 2 of 2)

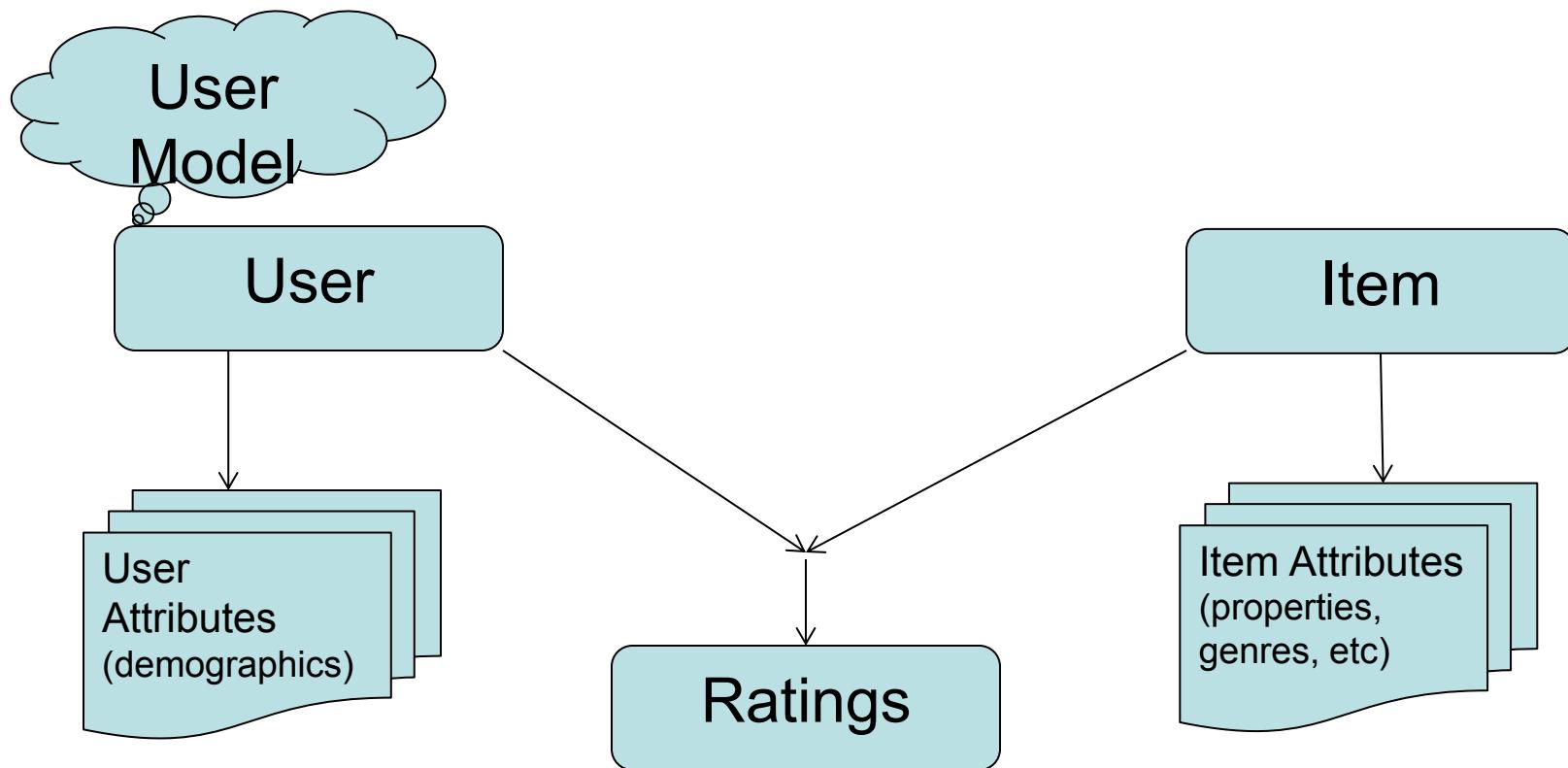
# Recommendation Algorithms

- Non-Personalized Summary Statistics
- Content-Based Filtering
  - Information Filtering
  - Knowledge-Based
- Collaborative Filtering
  - User-User
  - Item-Item
  - Dimensionality Reduction
- Others
  - Critique / Interview Based Recommendations
  - Hybrid Techniques

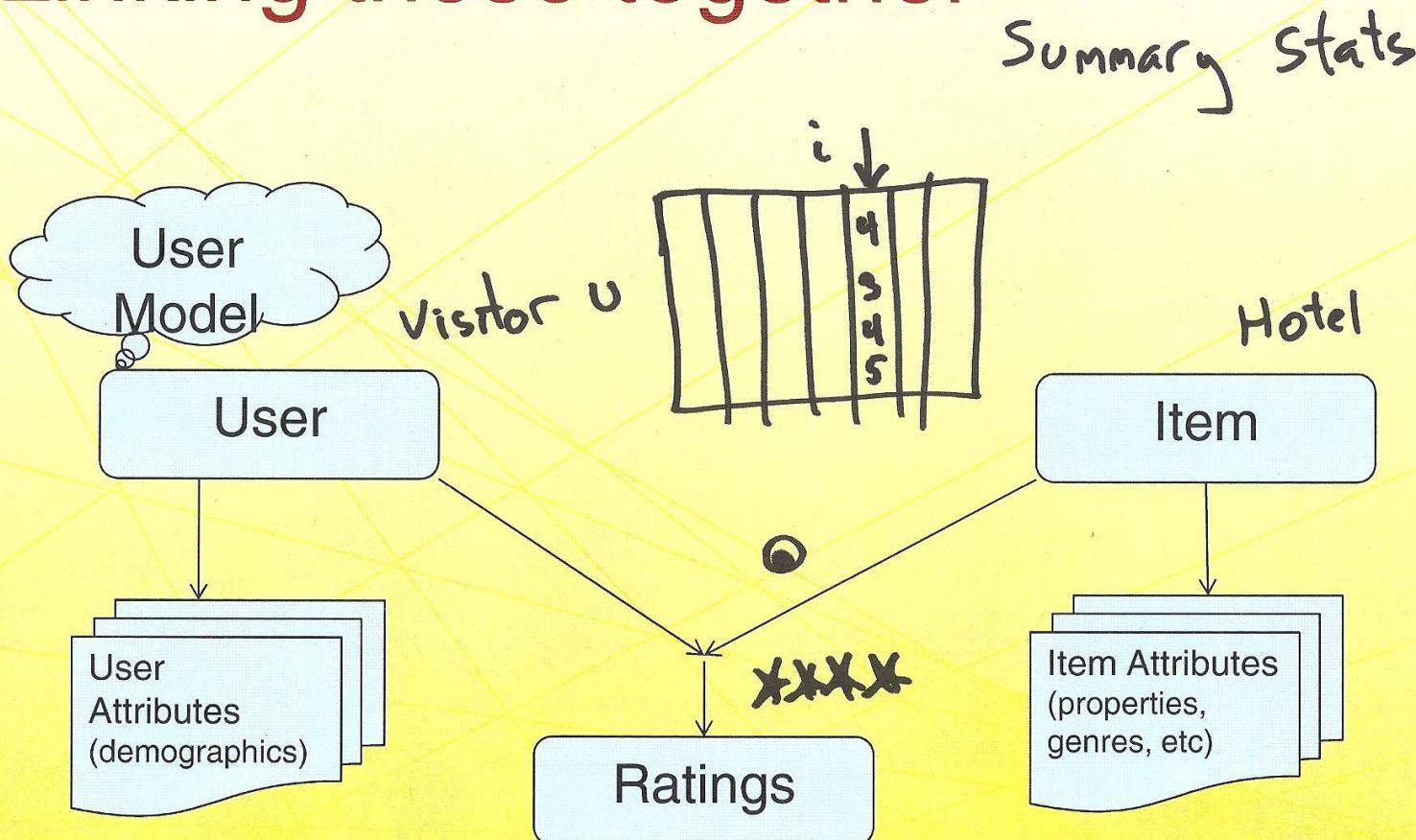
# From the Abstract to the Specific

- Basic Model
  - Users
  - Items
  - Ratings
  - (Community)

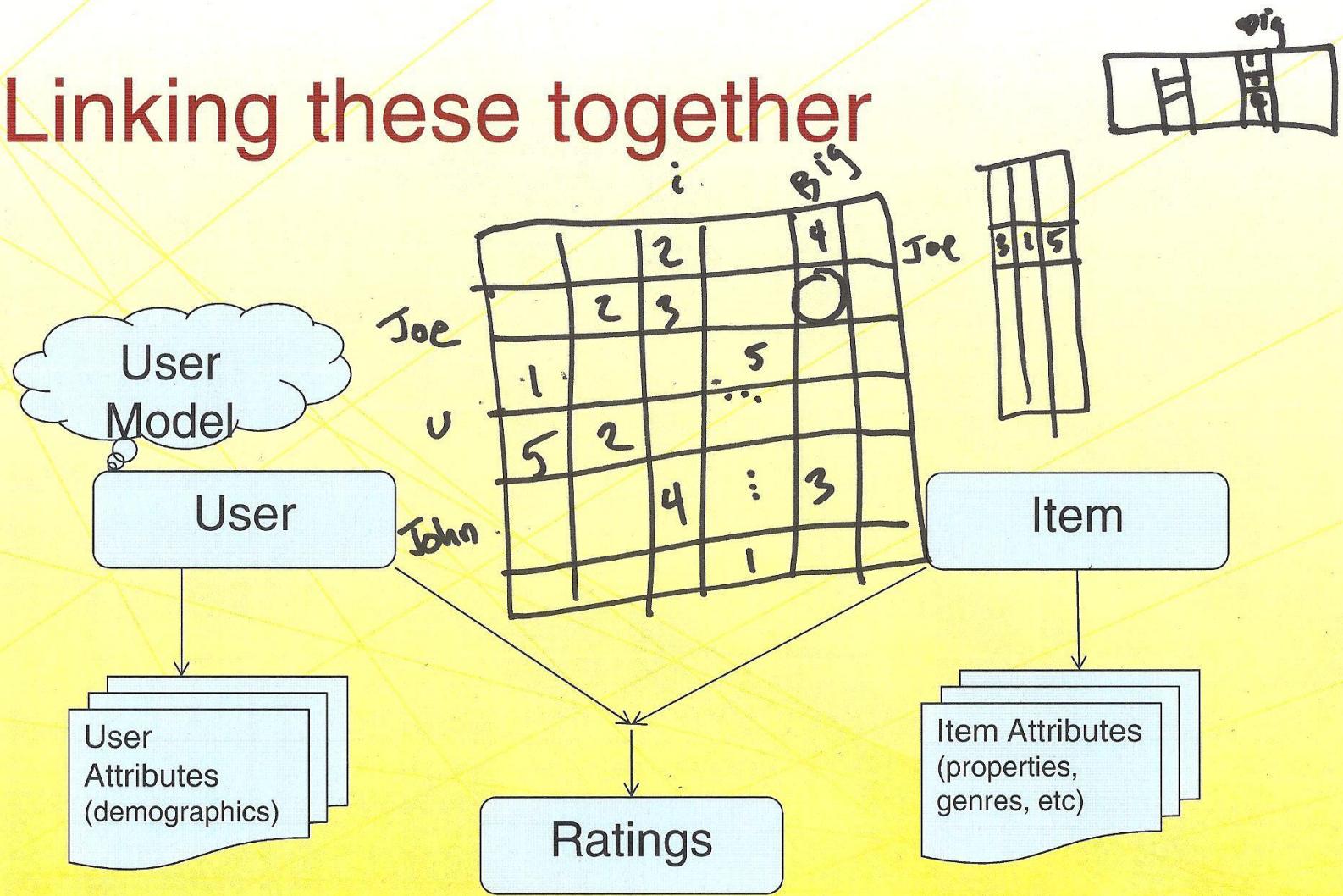
# Linking these together



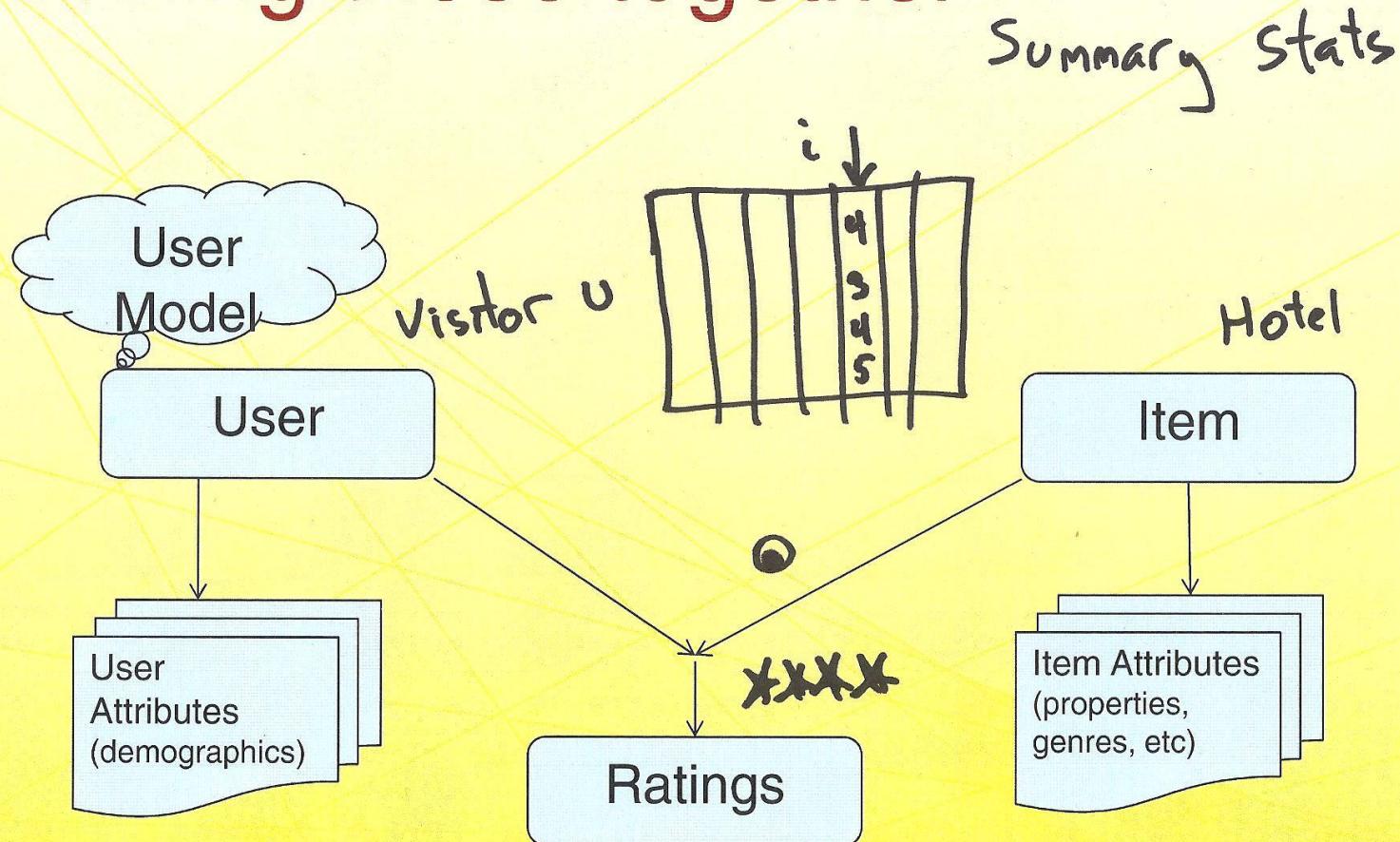
# Linking these together



# Linking these together



# Linking these together



# Non-Personalized Summary Stats

- External Community Data
  - Best-seller; Most popular; Trending Hot
- Summary of Community Ratings
  - Best-liked
- Examples
  - Zagat restaurant ratings
  - Billboard music rankings
  - TripAdvisor hotel ratings

# Content-Based Filtering

- User Ratings x Item Attributes => Model
- Model applied to new items via attributes
- Alternative: knowledge-based
  - Item attributes form model of item space
    - Users navigate/browse that space
- Examples
  - Personalized news feeds
  - Artist or Genre music feeds

# Personalized Collaborative Filtering

- Use opinions of others to predict/recommend
- User model – set of ratings
- Item model – set of ratings
- Common core: sparse matrix of ratings
  - Fill in missing values (predict)
  - Select promising cells (recommend)
- Several different techniques

# Collaborative Filtering Techniques

- User-user
  - Select neighborhood of similar-taste people
    - Variant: select people you know/trust
  - Use their opinions
- Item-item
  - Pre-compute similarity among items via ratings
  - Use own ratings to triangulate for recommendations
- Dimensionality reduction
  - Intuition: taste yields a lower-dimensionality matrix
  - Compress and use a taste representation

# Note on Evaluation

- To properly understand relative merits of each approach, we will spend significant time on evaluation
  - Accuracy of predictions
  - Usefulness of recommendations
    - Correctness
    - Non-obviousness
    - Diversity
  - Computational performance

# Other Approaches

- Interactive recommenders
  - Critique-based, dialog-based
- Hybrids of various techniques

# Moving Forward

- Next Lecture: A Tour of Amazon.Com, organized by our taxonomy
- Then, you should be able to:
  - Analyze a recommender application on your own
- Course Structure:
  - We step through the recommendation algorithms, with six major modules
  - Related topics intermingled

# 1-4: Taxonomy of Recommender Systems

# 1-5: A Tour of Amazon.com (with notes on review assignment)

# Learning Objectives

- To explore a wide range of recommender systems in the context of a large, professional site
- To understand how to review a recommender-enabled site

# Analytical Framework

- Dimensions of Analysis
  - Domain
  - Purpose
  - Recommendation Context
  - Whose Opinions
  - Personalization Level
  - Privacy and Trustworthiness
  - Interfaces
  - Recommendation Algorithms

# Analytical Framework

- Dimensions of Analysis
  - Domain
  - Purpose
  - Recommendation Context
  - Whose Opinions
  - Personalization Level
  - Privacy and Trustworthiness
  - Interfaces
  - Recommendation Algorithms

# Analytical Framework

- Dimensions of Analysis
  - Domain
  - Purpose
  - Recommendation Context
  - Whose Opinions
  - Personalization Level
  - Privacy and Trustworthiness
  - Interfaces
  - Recommendation Algorithms

# Analytical Framework

- Dimensions of Analysis
  - Domain
  - Purpose
  - Recommendation Context
  - Whose Opinions
  - Personalization Level
  - Privacy and Trustworthiness
  - Interfaces
  - Recommendation Algorithms

# Analytical Framework

- Dimensions of Analysis

- Domain *Products*
- Purpose *sales*
- Recommendation Context *time // product-associated*
- Whose Opinions *other customers (Phoaks) // people who looked at/bought x*
- Personalization Level *NONE // ephemeral*
- Privacy and Trustworthiness *low risk*
- Interfaces *explain + products*
- Recommendation Algorithms *aggregation / product assoc. + business rules*

# Analytical Framework

- Dimensions of Analysis
  - Domain
  - Purpose
  - Recommendation Context
  - Whose Opinions
  - Personalization Level
  - Privacy and Trustworthiness
  - Interfaces
  - Recommendation Algorithms

# Analytical Framework

- Dimensions of Analysis
  - Domain
  - Purpose
  - Recommendation Context
  - Whose Opinions
  - Personalization Level
  - Privacy and Trustworthiness
  - Interfaces – transparent
  - Recommendation Algorithms

unKnown

persistent // based on historical purchases

& ratings

Non-pers  
summary  
item-item  
c.f.

&  
browsing



Your Amazon.com | Today's Deals | Gift Cards | Sell | Help

Celebrate 4<sup>th</sup> of July [Shop now](#)

Shop by Department

Search

All

Go

Hello. Sign in  
Your Account

Join Prime



0

Cart

Wish List

Instant Video

MP3 Store

Cloud Player

Kindle

Cloud Drive

Appstore  
for AndroidDigital Games  
& SoftwareAudible  
Audiobooks

## kindle fire HD

Save \$30 on the ultimate HD experience

From  
~~\$199~~ \$169 [Shop now](#)

Limited-time offer



A Good Night's Sleep

Clothing Trends

Fast Free Shipping

## Amazon Fashion

# Summer Dresses

The latest sunny styles from  
Plenty by Tracy Reese and more.

[Shop Dresses](#)[Shop All Clothing](#)

## What Other Customers Are Looking At Right Now

Philips AS140/37 Fidelio  
Bluetooth...

★★★★★ (126)  
\$109.99 **\$54.49**



Kindle Fire HD 7", Dolby  
Audio...  
★★★★★ (14,015)  
\$149.00 **\$169.00**



Xbox One Console - Day One  
Edition  
Microsoft  
Xbox One  
\$499.96



Canon PowerShot SX260 HS  
12.1 MP CMOS...  
★★★★★ (847)  
\$340.00 **\$231.95**



Pilot [HD]  
Amazon Instant Video  
★★★★★ (130)  
\$2.99

DC Cupcakes

TLC's DC Cupcakes now available

Unlimited access to thousands of TV shows

[Start Free Trial](#)

amazon

Advertisement

## Pre-order the new consoles



XBOX ONE

PS4

[Learn more](#)

## Celebrate July 4<sup>th</sup>

with Deals on Computers &amp; Accessories

[Shop now](#)

= **\$30 Off**  
Instantly

[Learn more](#)

★★★★★ (126)  
\$100.99 \$54.49

★★★★★ (14,015)  
\$100.00 \$169.00

MICROSOFT  
Xbox One  
\$499.96

★★★★★ (847)  
\$340.00 \$231.95

★★★★★ (130)  
\$2.99

## Digital Cameras Best Sellers



Canon PowerShot A2300  
16.0 MP Digital...

★★★★★ (695)  
\$139.00 \$79.95



Canon PowerShot SX500 IS  
16.0 MP...

★★★★★ (194)  
\$299.00 Click for details



Foscam FI8910W Pan & Tilt  
IP/Network...

★★★★★ (2,758)  
\$100.95 \$84.95



Canon PowerShot SX50 HS  
12MP Digital...

★★★★★ (327)  
\$449.00 Click for details



Canon EOS Rebel T3i 18 MP  
CMOS...

★★★★★ (812)  
\$649.00 Click for details



Unlimited access to  
thousands of TV shows  
Prime Instant Video

Start Free Trial  
**amazon**

Advertisement

> See all best sellers in Digital Cameras

## Get Charged with Powerful and Versatile Equipment



[Emergency & Portable Power](#)



[Solar Chargers](#)



[Wind Power](#)

> All jump starters, chargers, and portable power

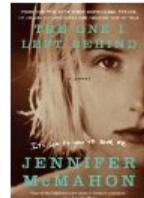
## Your Recent History

(What's this?)

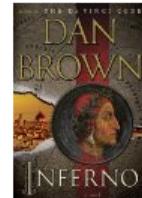
You have no recently viewed items.

After viewing product detail pages, look here to find an easy way to navigate back to pages you are interested in.

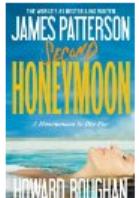
### Continue Shopping: Top Sellers



[The One I Left Behind](#)  
Jennifer McMahon



[Inferno: A Novel](#) (Robert Langdon)



[Second Honeymoon](#)  
James Patterson



[Damaged 2](#)  
H.M. Ward



[Surrender Your Love](#)  
J.C. Reed

Page 1 of 10

a Amazon.com

www.amazon.com/s/ref=nb\_sb\_noss\_1?url=search-alias%3Daps&field-keywords=ice+20maker&sprefix=ice+m%2Caps&rh=i%3Aaps%2Ck%3Aice%20maker

amazon Join Prime Your Amazon.com | Today's Deals | Gift Cards | Sell | Help

Celebrate 4<sup>th</sup> of July >[Shop now](#)

Shop by Department All ice maker Go

Hello, Sign in Your Account | Join Prime | Cart (0) | Wish List

**Departments**

Appliances

Refrigerator Replacement Ice Makers

Ice Makers

+ See more...

Kitchen & Dining

Ice Cream Machines

Kitchen Small Appliances

+ See more...

Patio, Lawn & Garden

Outdoor Ice Machines

Industrial & Scientific

Lab Ice Makers

+ See All 29 Departments

Shipping Option ([What's this?](#))

Free Super Saver Shipping

Brand

NewAir

EdgeStar

Sunpentown

Whirlpool

Magic Chef

Supco

U-Line

+ See more...

Avg. Customer Review

★★★★★ & Up

★★★★★ & Up

★★★★★ & Up

★★★★★ & Up

International Shipping ([What's this?](#))

AmazonGlobal Eligible

Condition

New

Used

"ice maker"

Related Searches: [ice cream maker](#), [ice machine](#), [portable ice maker](#).

Showing 1 - 16 of 18,871 Results Choose a Department  to enable sorting

**NewAir AI-100R 28-Pound Portable Icemaker, Red**

\$271.36 **\$187.98** Prime

Order in the next **6 hours** and get it by Tuesday, Jul 2.

More Buying Choices  
**\$182.95** new (14 offers)

**EdgeStar Titanium Portable Ice Maker - Titanium**

**\$114.00**

In Stock

More Buying Choices  
**\$114.00** new (2 offers)

**NewAir AI-100BK 28-Pound Portable Ice Maker, Black**

\$271.36 **\$189.99** Prime

Only 18 left in stock - order soon.

More Buying Choices  
**\$189.95** new (10 offers)

**NewAir AI-100SS 28-Pound Portable Ice Maker, Stainless Steel**

\$200.93 **\$206.98** Prime

In Stock

More Buying Choices  
**\$202.45** new (12 offers)

Eligible for FREE Super Saver Shipping.

Appliances: See all 6,188 items

Eligible for FREE Super Saver Shipping.

Appliances: See all 6,188 items

Eligible for FREE Super Saver Shipping.

Appliances: See all 6,188 items

[www.amazon.com/NewAir-AI-100R-28-Pound-Portable-Icemaker/dp/B0017V3GGU/ref=sr\\_1\\_1?ie=UTF8&qid=1372682708&sr=8-1&keywords=ice+maker](http://www.amazon.com/NewAir-AI-100R-28-Pound-Portable-Icemaker/dp/B0017V3GGU/ref=sr_1_1?ie=UTF8&qid=1372682708&sr=8-1&keywords=ice+maker)

Amazon.com

www.amazon.com/s/ref=nb\_sb\_noss\_1?url=search-alias%3Daps&field-keywords=ice%20maker&sprefix=ice+m%2Caps&rh=i%3Aaps%2Ck%3Aice%20maker

amazon Join Prime

Your Amazon.com Today's Deals Gift Cards Sell Help

Celebrate 4<sup>th</sup> of July Shop now

Shop by Department Search All ice maker Go Hello Sign in Your Account Join Prime Cart 0 Wish List

Departments Appliances Refrigerator Replacement Ice Makers Ice Makers + See more... Kitchen & Dining Ice Cream Machines Kitchen Small Appliances + See more... Patio, Lawn & Garden Outdoor Ice Machines Industrial & Scientific Lab Ice Makers + See All 29 Departments Shipping Option (What's this?) Free Super Saver Shipping Brand NewAir EdgeStar Sunpentown Whirlpool Magic Chef Supco U-Line + See more... Avg. Customer Review ★★★★☆ & Up ★★★☆☆ & Up ★★☆☆☆ & Up ★☆☆☆☆ & Up International Shipping (What's this?) AmazonGlobal Eligible Condition New Used

"ice maker"

Related Searches: ice cream maker, ice machine, portable ice maker.

Showing 1 - 16 of 18,871 Results Choose a Department to enable sorting

NewAir AI-100R 28-Pound Portable Icemaker, Red  
\$274.36 \$187.98 Prime Order in the next 6 hours and get it by Tuesday, Jul 2 More Buying Choices \$182.95 new (14 offers)



EdgeStar Titanium Portable Ice Maker - Titanium  
\$114.00 In Stock More Buying Choices \$114.00 new (2 offers)



NewAir AI-100BK 28-Pound Portable Ice Maker, Black  
\$274.36 \$189.99 Prime Only 18 left in stock - order soon. More Buying Choices \$189.95 new (10 offers)



NewAir AI-100SS 28-Pound Portable Ice Maker, Stainless Steel  
\$299.93 \$206.98 Prime In Stock More Buying Choices \$202.45 new (12 offers)



Eligible for FREE Super Saver Shipping. Appliances: See all 6,188 items

Eligible for FREE Super Saver Shipping. Appliances: See all 6,188 items

Eligible for FREE Super Saver Shipping. Appliances: See all 6,188 items



Your Amazon.com | Today's Deals | Gift Cards | Sell | Help

Celebrate 4<sup>th</sup> of July [Shop now](#)

Shop by Department

Search

Appliances

ice maker

Go

Hello. Sign in Your Account

Join Prime

0 Cart

Wish List

Appliances Best Sellers Refrigeration Cooking Washers &amp; Dryers Dishwashers Parts &amp; Accessories Installations &amp; Services



Roll over image to zoom in

[Share your own customer images](#)

## Product Features

- Convenient, compact design is ideal for use in small kitchens & other compact spaces like RVs, boats & more
  - Produces 28 pounds of ice a day - never run to the store for a bag of ice again
  - Offers an easy-to-use LED control panel that allows you to choose from three ice size settings
  - Portable ice maker requires no installation - just plug in unit, add water & wait up to 15 minutes to enjoy fresh ice
  - Flawless design boasts an excellent reputation, offering consistent & dependable ice production that has been continually perfected
- [See more product details](#)

## Frequently Bought Together



Price for both: \$193.94

These items are shipped from and sold by different sellers. Show details

 This item: NewAir AI-100BK 28-Pound Portable Ice Maker, Black by NewAir \$189.99 Ice Scoop, Stainless Steel by Ice Scoop \$3.95

Quantity: 1

or

[Sign in](#) to turn on 1-Click ordering. 2-Year Warranty  
\$16.99

## More Buying Choices

Wayfair  
\$189.95 Beach Camera  
\$189.95 Air & Water Inc  
\$189.95 10 new from \$189.95  
[Have one to sell?](#)

Email Facebook Twitter Print



## Frequently Bought Together



+



Price for both: \$193.94

[Add both to Cart](#)[Add both to Wish List](#)These items are shipped from and sold by different sellers. [Show details](#)

**This item:** NewAir AI-100BK 28-Pound Portable Ice Maker, Black by NewAir \$189.99

Ice Scoop, Stainless Steel by Ice Scoop \$3.95

## Product Information

### Technical Details

|                    |                         |
|--------------------|-------------------------|
| Brand Name         | NewAir                  |
| Model Info         | AI-100BK                |
| Item Weight        | 24.3 pounds             |
| Product Dimensions | 14.5 x 11.8 x 15 inches |
| Item model number  | AI-100BK                |
| Installation Type  | counter top             |
| Part Number        | AI-100BK                |
| Color              | Black                   |
| Voltage            | 120 volts               |
| Ice storage        | 28                      |

### Additional Information

|                      |   |
|----------------------|---|
| ASIN                 | B0017Y8X3O  |
| Customer Reviews     | <a href="#">56 reviews</a><br>3.8 out of 5 stars  |
| Best Sellers Rank    | #6,955 in Appliances ( <a href="#">See top 100</a> )<br>#7 in Appliances > Ice Makers   |
| Shipping Weight      | 28.6 pounds ( <a href="#">View shipping rates and policies</a> )  |
| Shipping             | This item can only be shipped to the 48 contiguous states. We regret it cannot be shipped to APO/FPO, Hawaii, Alaska, or Puerto Rico. |
| Date First Available | October 2, 2003   |

### Feedback

Did we miss any relevant features for this product? Tell us what we missed.

Would you like to give feedback on images or tell us about a lower price?

## Customers Who Bought This Item Also Bought

Page 1 of 8



Ice Scoop, Stainless Steel

(44)

\$3.95



Duracell 813-0807 800 Watt DC to AC Digital Power Inverter

(16)



The Survival Medicine Handbook: A guide for ...

by Joseph Alton M.D.

(2)



Powerex MH-C800S Eight Cell Smart Charger

(81)

\$54.95



3/8" Fuel Primer Bulb

(6)

\$15.33



Fuel Line Hose, 3/8" x 25'

(13)

\$21.03

&lt; &gt;



Your Amazon.com | Today's Deals | Gift Cards | Sell | Help

Celebrate 4<sup>th</sup> of July >Shop now

Shop by Department

Search

Appliances

Go

Hello. Sign in  
Your Account

Join Prime



Wish List

Appliances Best Sellers Refrigeration Cooking Washers &amp; Dryers Dishwashers Parts &amp; Accessories Installations &amp; Services

## ✓ 1 item added to Cart

[NewAir AI-100BK 28-Pound Portable Ice Maker, Black](#)

\$189.99

Only 18 left in stock.

 Protect this product with a 2-year warranty \$16.99 This will be a gift

Order subtotal: \$189.99

1 item in your Cart

[Edit your Cart](#)[Proceed to checkout](#)

NewAir AI-100BK 28-Pound Portable Ice Maker, Black

Your order qualifies for free shipping!

Select FREE Super Saver Shipping at checkout. (Some restrictions apply)

**6**  
Month  
Financing

Your cart is eligible for a financing offer

**6 Month Special Financing on orders \$149 or more**[Apply now](#)with the **Amazon.com Store Card**. See details and restrictions.

### Recommended for You Based on NewAir AI-100BK 28-Pound Portable Ice Maker, Black

**NewAir AI-100R 28-Pound Portable Icemaker, Red**

(116)

\$271.36 **\$187.98**

14 New from \$182.95

[Add to Cart](#)**Ice Scoop, Stainless Steel**

(44)

\$3.95

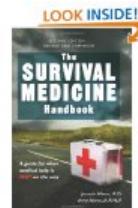
13 New from \$0.01

[Add to Cart](#)**NewAir AI-100S 28-Pound Portable Ice Maker, Silver**

(105)

\$271.36 **\$187.98**

14 New from \$179.95

[Add to Cart](#)**The Survival Medicine Handbook**by Joseph Alton M.D.  
Paperback

(2)

\$39.99 **\$33.67**[Add to Cart](#)

### Customers Who Shopped for NewAir AI-100BK 28-Pound Portable Ice Maker, Black Also Shopped For





### Customers Also Bought these Highly Rated Items



[Duracell 813-0807 800 Watt DC to AC...](#)

★★★★★ (16)

\$59.99 \$47.00  
37 New from \$47.00

[Add to Cart](#)



[Powerex MH-C800S Eight Cell Smart Charger](#)

★★★★★ (81)

\$64.95 \$54.95  
11 New from \$54.00

[Add to Cart](#)



[Fuel Line Hose, 3/8" x 25'](#)

★★★★★ (13)

\$26.99 \$21.03  
8 New from \$16.41

[Add to Cart](#)



[OXO Good Grips Scoop, Translucent White](#)

★★★★★ (44)

\$5.99  
7 New from \$5.89

[Add to Cart](#)

### Customers Who Bought *NewAir AI-100BK 28-Pound Portable Ice Maker, Black* Also Bought These Items from Other Categories



[3/8" Fuel Primer Bulb](#)

★★★★★ (6)

\$15.33  
6 New from \$8.25

[Add to Cart](#)



[Lights of America 2326LED-LF4-24 2-Watt Power LED...](#)

★★★★★ (67)

\$10.99 \$5.50  
7 New from \$5.25

[Add to Cart](#)



[PRI Fuel Stabilizer- For Gasoline 32oz](#)

★★★★★ (7)

\$35.49  
4 New from \$30.95

[Add to Cart](#)



[SANYO NEW 1500 eneloop 8 batteries AAA...](#)

★★★★★ (130)

\$19.96  
7 New from \$12.99

[Add to Cart](#)

### Your Recent History (What's this?)

#### Recently Viewed Items



NewAir AI-100BK 28-Pound...  
NewAir

#### Continue Shopping: Customers Who Bought Items in Your Recent History Also Bought



NewAir AI-100S 28-Pound Portable



NewAir AI-100R 28-Pound Portable



Ice Scoop, Stainless Steel



Duracell 813-0807 800 Watt DC to...

Page 1 of 11



Do you want Google Chrome to save your password?



Joseph's Amazon.com | Today's Deals | Gift Cards | Sell | Help

Celebrate 4<sup>th</sup> of July [Shop now](#)

Shop by Department

Search

All

Go

Hello, Joseph  
Your Account

Your Prime

3 Cart

Wish List

Your Amazon.com Your Browsing History Recommended For You Amazon Betterizer Improve Your Recommendations Your Profile Learn More

## Your Amazon.com

Featured Recommendations

Electronics

Health & Personal Care

Cell Phones & Accessories

Office Products

Home Improvement

Toys & Games

See All Recommendations

### Electronics

Page 1 of 17



SANOXY MINI USB FEMAL...  
 (99)  
\$1.70  
[Why recommended?](#)



StarTech.com 6 Inch M...  
 (131)  
\$25.48 **\$4.75**  
[Why recommended?](#)



Monoprice 3ft USB 2.0...  
 (4)  
\$19.99 **\$3.77**  
[Why recommended?](#)



StarTech 1ft Micro US...  
 (286)  
\$13.00 **\$3.74**  
[Why recommended?](#)



Simran SMF-200 Deluxe...  
 (496)  
\$14.99 **\$13.49**  
[Why recommended?](#)



Tripp Lite U050-003 3...  
 (117)  
\$16.00 **\$3.39**  
[Why recommended?](#)



[See all recommendations in Electronics](#)

Page 1 of 17

### Health & Personal Care



Glad Tall Kitchen Dra...



Truefitt & Hill 1805...



Quilted Northern Ultr...



Sparkle Paper Towels...



Angel Soft, Double Ro...



Puffs Plus Lotion Fac...





Joseph's Amazon.com | Today's Deals | Gift Cards | Sell | Help

Celebrate 4<sup>th</sup> of July [Shop now](#)

Shop by Department

Search

All

Go

Hello, Joseph  
Your Account

Your Prime



Wish List

Your Amazon.com | Your Browsing History | Recommended For You | Amazon Betterizer | Improve Your Recommendations | Your Profile | Learn More

## Your Amazon.com &gt; Recommended for You

(If you're not Joseph A. Konstan, click here.)

## Just For Today

[Browse Recommended](#)

## Recommendations

[Amazon Instant Video](#)[Amazon MP3 Store](#)[Appliances](#)[Appstore for Android](#)[Arts, Crafts & Sewing](#)[Automotive](#)[Baby](#)[Beauty](#)[Books](#)[Books on Kindle](#)[Camera & Photo](#)[Cell Phones & Accessories](#)[Clothing & Accessories](#)[Computers](#)[Electronics](#)[Grocery & Gourmet Food](#)[Health & Personal Care](#)[Home & Kitchen](#)[Home Improvement](#)[Industrial & Scientific](#)[Jewelry](#)[Kitchen & Dining](#)[Magazine Subscriptions](#)[Magazines on Kindle](#)[Movies](#)[Movies & TV](#)[Music](#)[Musical Instruments](#)[Newspapers on Kindle](#)[Office & School Supplies](#)These recommendations are based on [items you own](#) and more.view: [All](#) | [New Releases](#) | [Coming Soon](#)[More results](#)

1.

[Beyblades #BB106 JAPANESE Metal Fusion Starter Set Fang Leone 130W2D](#)

by Takaratomy (March 28, 2011)

Average Customer Review: (49)

In Stock

**List Price:** \$15.99**Price:** \$8.73

69 new from \$5.99

[Add to Cart](#)[Add to Wish List](#) I own it  Not interested  Rate this itemRecommended because you added [Beyblades JAPANESE Metal Fusion Starter Set #BB105 Big Ba...](#) to your Wishlist and more ([Fix this](#))

2.

[Beyblades JAPANESE Metal Fusion Starter Set #BB108 LDrago Destroy F:S](#)

by Takaratomy (April 23, 2011)

Average Customer Review: (43)

In Stock

**List Price:** \$15.99**Price:** \$7.37

69 new from \$5.99

[Add to Cart](#)[Add to Wish List](#) I own it  Not interested  Rate this itemRecommended because you added [Beyblades JAPANESE Metal Fusion Starter Set #BB105 Big Ba...](#) to your Wishlist and more ([Fix this](#))

3.

[eForCity USB 2.0 A to Micro B Female / Male Adaptor](#)

by Generic (May 10, 2004)

Average Customer Review: (96)

In Stock

**List Price:** \$7.99**Price:** \$1.02

41 used &amp; new from \$0.01

Offered by [ExquisiteGadgetsWorld](#)[Add to Cart](#)[Add to Wish List](#) I own it  Not interested  Rate this itemRecommended because you purchased [eForCity Micro USB OTG to USB 2.0 Adapter](#) and more ([Fix this](#))

4.

[Google Nexus 4 \(LG E960\) Bumper Case - Black](#)

by Abacus24-7 (January 15, 2013)

https://www.amazon.com/gp/yourstore/iyr?ie=UTF8&ref\_=pd\_ys\_hn\_iyr\_own

Joseph's Amazon.com | Today's Deals | Gift Cards | Sell | Help
Celebrate 4<sup>th</sup> of July >[Shop now](#)

amazon Prime
Shop by Department
Search All ▾
Go
Hello, Joseph Your Account
Your Prime
 Cart
Wish List

Your Amazon.com Your Browsing History Recommended For You Amazon Betterizer Improve Your Recommendations Your Profile Learn More

## Your Amazon.com > Improve Your Recommendations

(If you're not Joseph A. Konstan, click here.)

Help us make better recommendations. You can refine your recommendations by rating items or adjusting the checkboxes.

**EDIT YOUR COLLECTION**

- [Items you've purchased](#)
- [Items you've marked "I own it"](#)
- [Items you've rated](#)
- [Items you've liked](#)
- [Items you've marked "Not interested"](#)
- [Items you've marked as gifts](#)

**EDIT YOUR PREFERENCES**

- Show Amazon book recommendations as Kindle editions when possible.

**Need Help?**  
Visit our [help](#) area to learn more.

### Items you've purchased

---

1.  [Geo. F. Trumper Limes Shaving Cream Tube](#)  
by Geo F. Trumper

**Your tags:**  [Add](#) (What's this?)  
[Click to Add: geo f trumper, mens, healthy living](#)

This was a gift
 Don't use for recommendations

---

2.  [Maxell LR6 AA Cell 48 Pack Box Battery \(723443\)](#)  
by Maxell

**Your tags:**  [Add](#) (What's this?)  
[Click to Add: alkaline batteries, aa batteries, alkaline, good value, cheap, value pack, aa](#)

This was a gift
 Don't use for recommendations

---

3.  [Hasty-Bake Gourmet 257 Stainless Steel Charcoal Grill](#)  
by Hasty-Bake

**Your tags:**  [Add](#) (What's this?)  
[Click to Add: hasty-bake, freestanding\\_grills](#)

This was a gift
 Don't use for recommendations

---

4.  [3M Compact Mouse Pad with Gel Wrist Rest, Black Leatherette, Antimicrobial Product Protection \(MW309LE\)](#)  
by 3M

**Your tags:**  [Add](#) (What's this?)  
[Click to Add: wrist rest, mousepad, mouse pad, ergonomic, laser mouse, mousepads, mouse, wrist rests](#)

This was a gift
 Don't use for recommendations



Joseph's Amazon.com | Today's Deals | Gift Cards | Sell | Help

Celebrate 4<sup>th</sup> of July [Shop now](#)

Shop by Department

Search

All

Go

Hello, Joseph  
Your Account

Your Prime



Wish List

Today's Deals | Gold Box | All Deals | Coupons | Outlet | Deals &amp; Bargains | Warehouse Deals | Digital Deals

Handles even your heaviest, messiest meals  
©2013 Dixie Consumer Products LLC. All rights reserved.Save \$1.00 On Dixie® Products  
(Discount at checkout)[Clip coupon](#)

Advertisement



## Gold Box New Deals. Every Day.

Never miss another deal 

## Deal of the Day

## 45% Off Puma BioWeb Elite Running Shoes



Save 45% on Puma BioWeb Elite Running Shoes for women and men. This running and training shoe is engineered to deliver maximum cushioning and stability—plus it comes in a wide range of sporty colors.

List Price: \$100.00

Yesterday's Price: \$77.98

Today's Discount: - \$23.03

Gold Box Price: \$54.95 (45% off)

7 Comments

[Learn more](#)[Share](#)

## Lightning Deals

1 - 4 of 11

[+] 7:00 PM PDT - Kitchen & Dining Deal [All Discounts Claimed](#)[+] 12:00 AM PDT - MP3 Players & Accessories Deal [Available](#)

## Philips Fidelio Bluetooth Speaker with Micro-USB Dock



List Price: \$109.99

Amazon's Price: \$54.49

Gold Box Discount: -\$4.50

Deal Price: \$49.99 (55% off)

Comments | (126)

[Add to Cart](#)

00:08:11 remaining

Prime  
Restrictions apply.[+] 6:00 AM PDT - Musical Instruments Deal [Coming Soon!](#)[+] 8:00 AM PDT - Upcoming Deal [Coming Soon!](#)Celebrate 4<sup>th</sup> of July [Shop now](#)kindle fire HD  
From \$199 ~~\$199~~ \$169 [Shop now](#)  
Limited-time offerFree Amazon Mobile App  
Shop millions of products wherever you go  
[Learn more](#)

## Best Deals &gt; More of our best deals

Sort by Original Order

Page 1 of 80

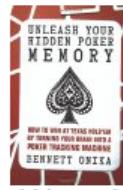


## Amazon Local Deals

**From \$65 (Save 54%)**Four-Course Prix Fixe French...  
Faces Mears Park[View Deal](#)St. Paul  
6 days**From \$35 (Save 71%)**Custom Canvas Prints and...  
CanvasFocus[View Deal](#)St. Paul  
1+ weeks**From \$13 (Save 75%)**Custom 20-Page Hardcover Photo...  
Picaboo[View Deal](#)St. Paul  
1+ weeks**\$15 (Save 50%)**\$30 to Spend on Fresh Summer...  
1800Flowers.com®[View Deal](#)St. Paul  
5 days**\$30 (Save 85%)**Auto Maintenance Package  
Auto Care Special[View Deal](#)St. Paul  
6 days

Page 1 of 9

**Joseph's Quick Picks** - To take advantage of the extra discount on one of these products chosen especially for you, add one item from this section to your cart and buy within an hour. New items added every day. [Learn more](#)



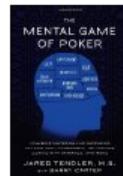
**Unleash Your Hidden Poker Memory:  
How to Win at...**

Going beyond the common... [read more](#)**★★★★★** (13)\$24.95 **\$18.84** (24% off)**Prime**[Add to Cart](#)

**The Math of Hold'em**

Winning big at holdem requires... [read more](#)**★★★★★** (2)\$34.95 **\$26.52** (24% off)**Prime**[Add to Cart](#)

**Phil Gordon's Little Gold Book:  
Advanced Lessons for...**

Since reigning poker expert Phil... [read more](#)**★★★★★** (19)\$25.00 **\$19.30** (23% off)**Prime**[Add to Cart](#)

**The Mental Game of Poker: Proven  
Strategies for...**

The mental game may be more



amazon led



konstan@umn.edu



2

+ Share



Mail



More

11 of many



COMPOSE

## Joseph A. Konstan: 30% or More Off Computer Monitors



Related Google+ Page

Amazon.com

Follow



Inbox (3)  
Sent Mail  
Drafts (24)  
Spam (1,139)  
#todo

Academic-Admin  
SocComp

CHI 2012

E-mail

Keynote

To Do

Classes

CSci 1001

CSci 5115

Project-Cont...

CSci 5116

► CSci 5125 (399)

CSci 5129

► CSci 8001 (5)

► CSci 8115 (2)

► CSci5115

e-Pub-Health

PhD

recsys

SEng5115-S04

SW5115-F00

Tutorials



Search people...

Ellen Konstan

Loren Terveen

Max Harper

Michael Ekstrand

<http://blab.umn.edu>

Amazon.com <store-news@amazon.com>

May 23 ★



Your Amazon.com | Today's Deals | See All Departments

Customers who have shown an interest in monitors might like to know about saving 30% or more on select computer monitors.



[Viewsonic VX2450WM-LED 24-Inch Widescreen LED Monitor with Full HD...](#)

[Learn more](#)

by ViewSonic

List Price: \$368.00

Price: \$189.99

You Save: \$178.01 (48%)



[ViewSonic VX2250WM-LED 22-Inch Widescreen Full HD 1080p LED...](#)

[Learn more](#)

by ViewSonic

List Price: \$279.99

Price: \$156.29

You Save: \$123.70 (44%)



[ViewSonic VA2406M-LED 24-Inch Screen LED-Lit Monitor](#)

[Learn more](#)

by ViewSonic

List Price: \$260.00

Price: \$159.99

You Save: \$100.01 (38%)

# Moving Forward

- Next Assignment
  - You will find and submit a review of a site that uses at least two different recommenders
  - You should classify your site by our taxonomy, including your best guess about algorithms
- Next Module: Non-Personalized
  - We will look at both summary statistics and product association (people who liked X) recommenders

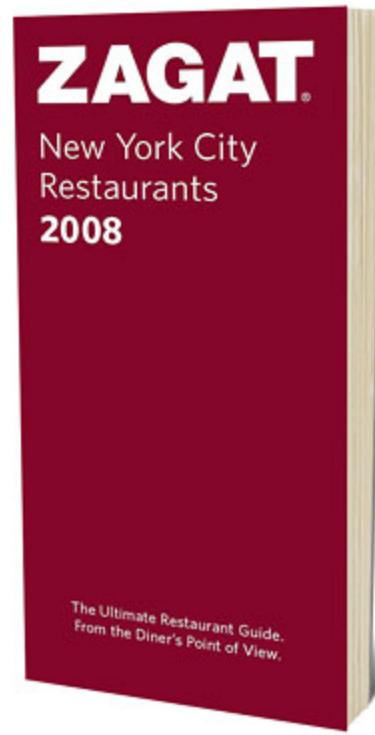
# 1-5: A Tour of Amazon.com (with notes on review assignment)

# 2-1: Introduction to Non-Personalized Recommenders

# Learning Objectives

- To understand the value of non-personalized recommenders, and domains where they are most useful
- To understand the drawbacks of non-personalized recommender systems
- To understand the basics of:
  - Aggregated opinion recommenders
  - Basic product association recommenders
- Review examples of the above ...

# The Story of Zagat



# The Zagat Guide ...



hail the new chef – Michael Anthony, formerly of Blue Hill at Stone Barns – and salute his “spellbinding” market-centric cuisine .....



**Gary Danko**   
San Francisco | American

| Food | Decor | Service | Cost  |
|------|-------|---------|-------|
| 29   | 26    | 28      | \$104 |

“Gary Swanko” “fully merits its superb reputation” gush “flush” “ies” who vote the “celebrity” chef-owner’s “sleek” New American “temple of gastronomy” in Fisherman’s Wharf No. 1 .....



**Charlie Trotter's**   
Chicago | American

| Food | Decor | Service | Cost |
|------|-------|---------|------|
| 27   | 25    | 27      | VE   |

“A religious experience” “worth a mortgage payment” awaits at Lincoln Park, the “epitome of [New] American gastronomy” at Chicagoland’s Most Popular restaurant, where customers .....



**Babbo**   
New York | Italian

| Food | Decor | Service | Cost |
|------|-------|---------|------|
| 27   | 25    | 27      | \$76 |

“When it’s this good” “it’s not hype” is still the consensus as Mario Batali and Joe Bastianich celebrate the 10th anniversary of their “fabulously popular” Village flagship that’s voted NYC’s No. 1 ...



**Spago**   
Los Angeles | Californian

| Food | Decor | Service | Cost |
|------|-------|---------|------|
| 27   | 25    | 25      | \$73 |

“Forget Gibraltar, this place is the rock of Los Angeles” sum up

# Secrets Revealed!

- The “secret” formula

Rating = {0, 1, 2, 3}

Score = round (MEAN(ratings) \* 10)

- OK, maybe not so secret – but effective!

CONDÉ NAST TRAVELER

CRUISE FINDER

CRUISE LINES

# Crystal Cruises

[CRUISE LINE WEBSITE](#)

|             |             |
|-------------|-------------|
| <b>94.5</b> | ITINERARIES |
| <b>88.4</b> | EXCURSIONS  |
| <b>97.7</b> | SERVICE     |
| <b>91.6</b> | CABINS      |
| <b>96.5</b> | FOOD        |
| <b>94.2</b> | ACTIVITIES  |
| <b>96.4</b> | DESIGN      |

[See Methodology »](#)
**READERS'  
CHOICE  
AWARDS**

Top 10  
Midsize-Ship  
Lines

**GOLD LIST**

2013

**THIS FLEET  
SAILS TO:**

South America

Its two ships' itineraries recently extended to Antarctica, Alaska, and West Africa. Excursions include local volunteer programs, helicopter rides in New Zealand, and splurges like a drive through Monte Carlo in a Lamborghini. On board, expect "the best lecturers and personalities of any cruise line." Rooms have "more than enough hanging and drawer space, and spectacular marble bathrooms"; Crystal Penthouses come with ocean view hot tubs, and some have Swarovski chandeliers. "The true standouts are the restaurants," which are all complimentary and include Silk Road and the Sushi Bar, a specialty dining room where the sushi is made by Nobu-trained chefs and "is beyond compare." Year to year, "staff remember your favorite cocktails." The spas use Elemis skin products, and there are Nordic Walking fitness classes.

**MOST POPULAR**
**MOST COMMENTED**

1. The Best New Bars Around the World
2. Airport Restaurants That Really Are Worth the Trip
3. We Dare You to Walk Across These Bridges
4. Patriotic Places that Will Make You Proud to Be an American
5. Artisanal Gelato: How to Spot the Fakes


**SEABOURN**  
**2014 WORLD CRUISE VOYAGES**  
  
[LEARN MORE](#)

 Like  134,801 people like this. Be the first of your friends.

 Follow @CNTraveler 356K followers

**Subscribe to The Daily Traveler  
Newsletter**



[SEE AN EXAMPLE](#) | [PRIVACY POLICY](#)

ADVERTISEMENT

**Silversea ® Cruises**

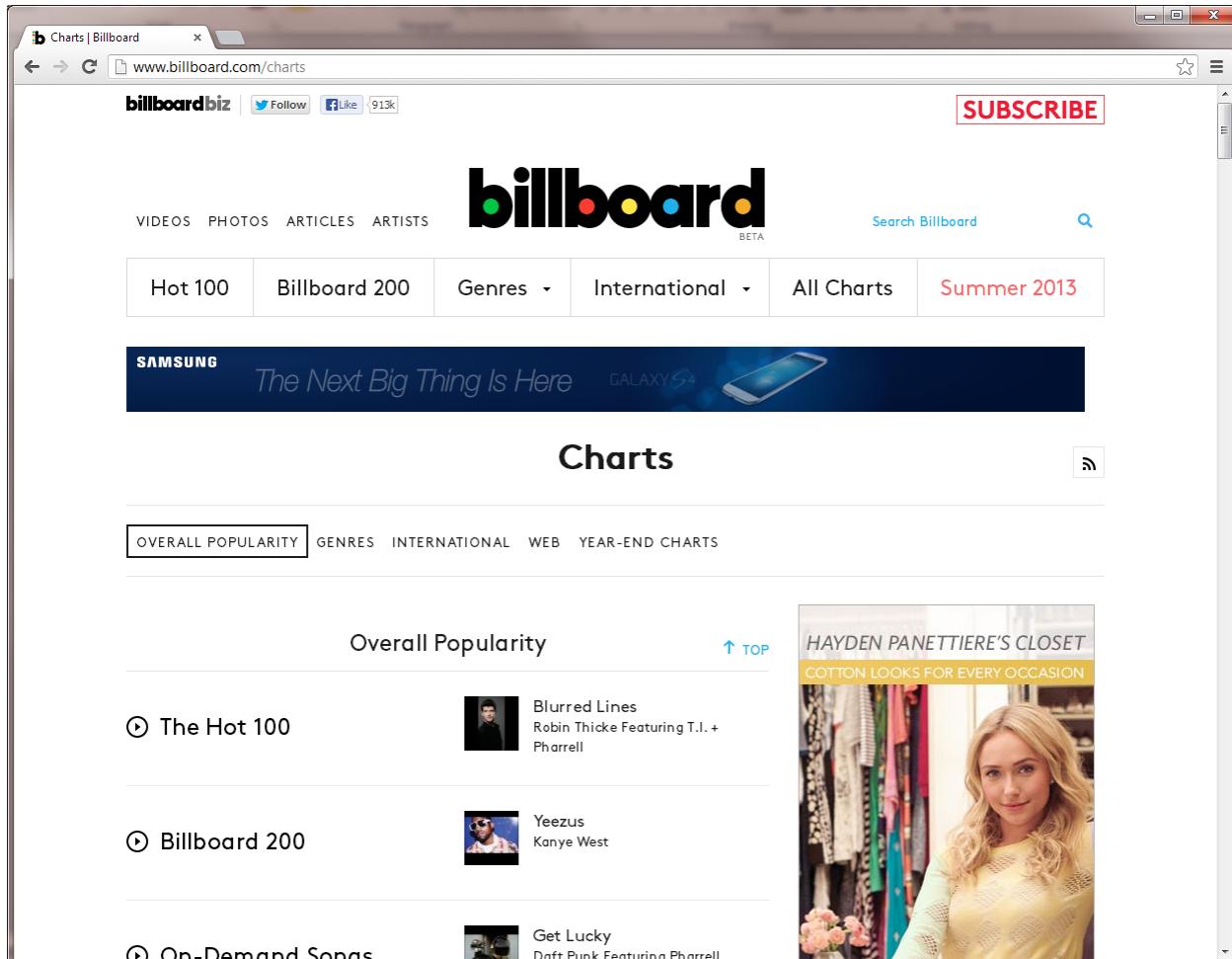
[www.Silversea.com/Official-Site](http://www.Silversea.com/Official-Site)

# Same idea, different formula

- Conde Nast Traveller tallies the percentage of people who rate a particular hotel, cruise, etc. as “very good” or “excellent”
- Relative merits of the two techniques ...
  - How do we treat a score of “good” vs. “awful”

# Many other examples

- Tripadvisor travel reviews and ratings
- Billboard top 200/100/20 ...
- Movie charts by box office revenue
- All non-personalized



# Averages can be Misleading

- Later this module ... we'll discuss ways to mislead using averages.
- See if you can come up with examples or ideas (post to the class forum, and vote up the ones you find most compelling)

# Averages Lack Context ...

- Ordering an ice-cream sundae
  - You want a recommendation for a sauce
  - Do you want to hear that ketchup is the most popular sauce?
- One interesting context is a current product (or set of products) – what sauce is most commonly associated with a sundae??
- This leads to the concept of product association recommenders!

# People who X also Y ...

- Great idea, but how to formalize
- First, what's our dataset
  - User profiles (people who ever bought one and the other)? – not good for ketchup
  - Transaction data (people who bought them at the same time)? – not good for follow-up sales
  - User profiles but time-constrained (within a month, afterwards, ...)?

# Computing the ranking

- Start simple: percentage of X-buyers who also bought Y

X and Y

---

X

- Intuitively right, but is it useful? What if X is anchovy paste and Y is bananas??
- Challenge – doesn't compensate for overall popularity of Y

# Take two – does X make Y more likely??

- Let's adjust by looking at whether X makes Y more likely than not  $X(!X)$

$X$  and  $Y$

$X$

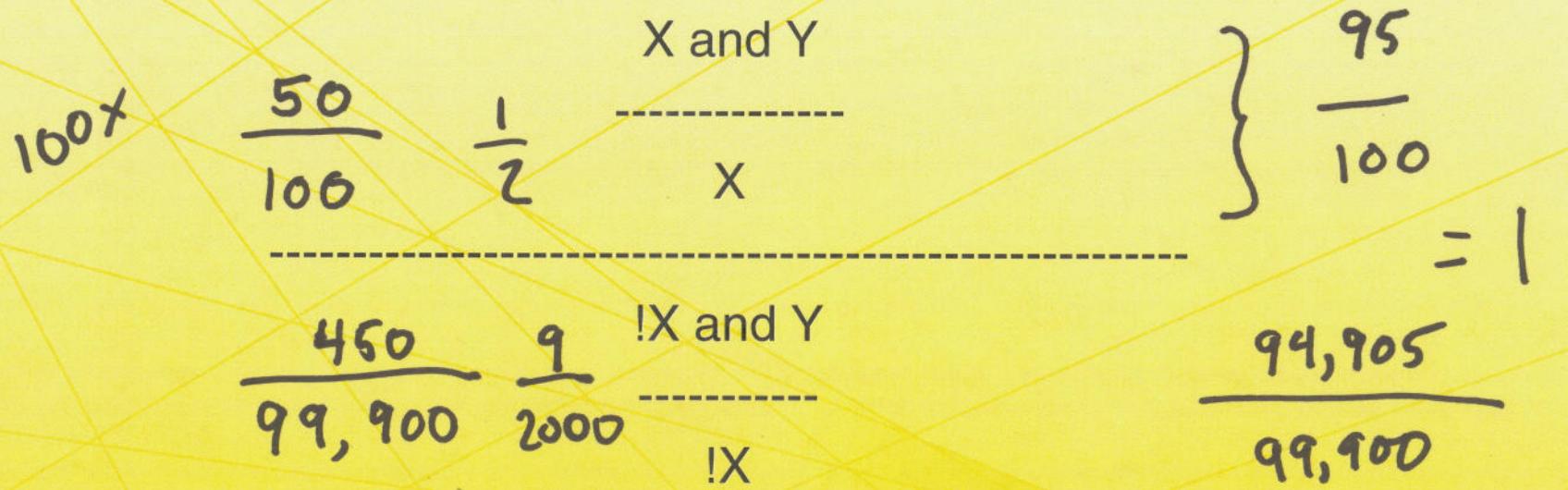
$X$  and  $Y$

$X$

- This formula focuses on increase in  $Y$  associated with  $X$

# Take two – does X make Y more likely??

- Let's adjust by looking at whether X makes Y more likely than not X(!X)



- This formula focuses on increase in Y associated with X

# Other solutions ...

- Association rule mining brings us the lift metric:

$$P(X \text{ AND } Y)$$

---

$$P(X) * P(Y)$$

- This looks at non-directional association
- More generally association rules look at baskets of products, not just individuals

# Back to Zagat

- Some early Zagat fans argue the guide has been getting worse. Why?
  - Too many mediocre restaurants with good scores
  - Too many excellent restaurants with mediocre scores
- What's happening here?
  - Self-selection bias
  - Increased diversity of raters

# Some take-away lessons

- Non-personalized averages can be effective in the right application
  - Need to understand relationship between average and user need; correct average
- Product associations can provide useful non-personalized recommendations in a context
  - Need to identify context; data source/scope
- Still face challenges in a clustered diverse population (e.g., maybe we don't all want bananas)

# Moving Forward

- Assignments this Module
  - Review an existing recommender
  - Hand-exercise: non-personalized recommender
  - Programming: non-personalized recommender
- Next lectures: about ratings, predictions and recommendations, rating scales
- Then, you should be able to:
  - Work out non-personalized recommendations
  - For programmers: program them too!

# 2-1: Introduction to Non-Personalized Recommenders

# 2-2: Preferences and Ratings

# Introduction

- To recommend, we need data (what users like, what goes together, etc.)
- Data comes from users, is collected somehow
- This lecture's topic: what data we collect, how, and what it means

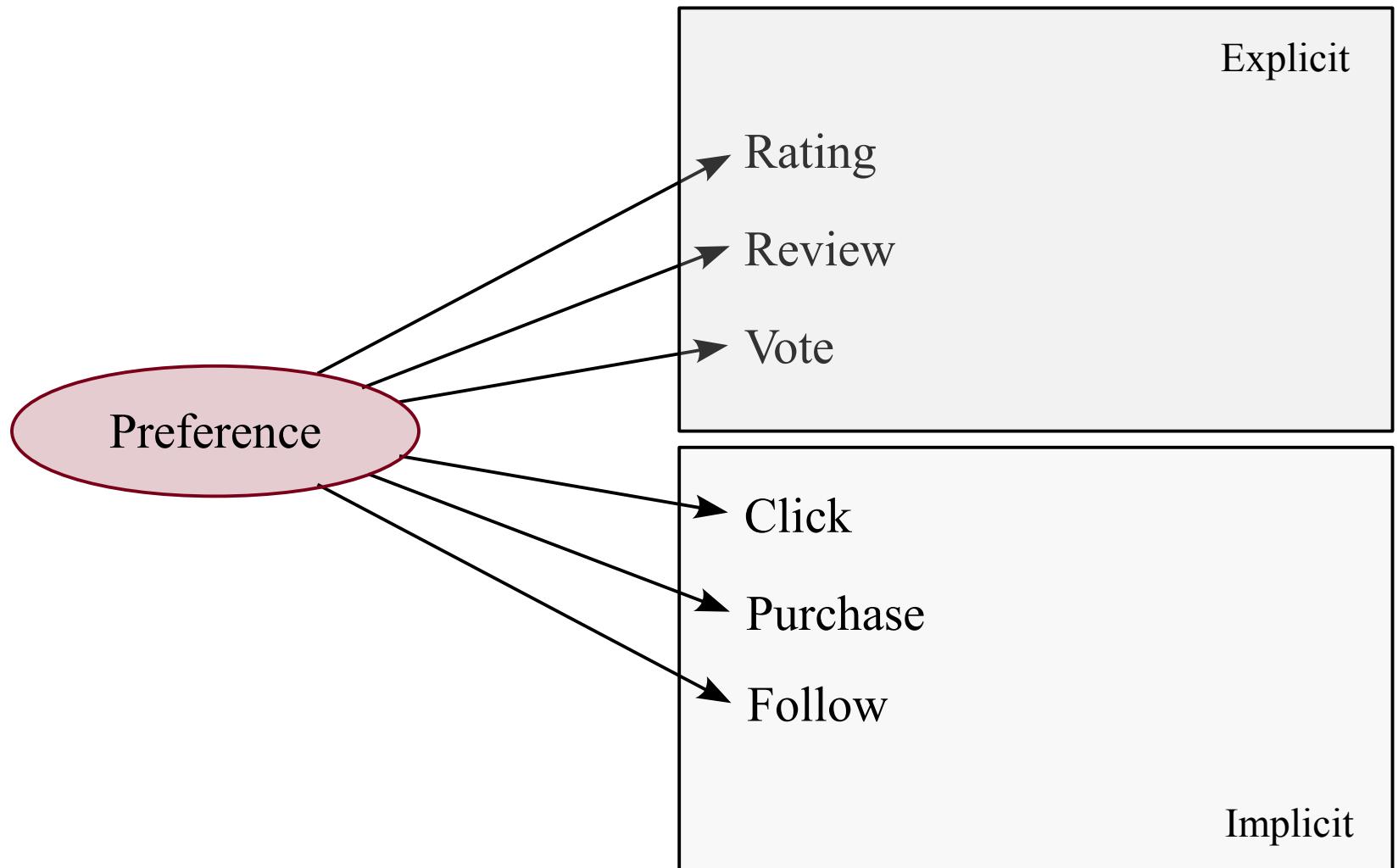
# Learning Objectives

- Understand what data recommenders can use to learn what users like
- Identify types of data collected from users
- Understand when different data types are possible and appropriate
- Be able to identify types of preference data likely used in a system

# Preference and Ratings

- We want to know: what do users like?
  - Or: what goes together?
- We can observe
  - What users tell us (ratings)
  - What users do (actions)
- These are *noisy measurements* of preference

# Preference Model



# Explicit Ratings

Just ask the users what they think!

# Star Ratings

- Widely-used interface
- Several design decisions
  - 5? 7? 10?
  - Half-stars?
  - Provide meaning/calibration?
  - More not necessarily better
- 5, with or without  $\frac{1}{2}$ , very common

**Sherlock**

2010-2012 TV-14 2 Series

In this updated take on Arthur Conan Doyle's beloved mystery tales, the eccentric sleuth prowls the streets of modern-day London in search of clues. [More Info](#)

**Starring:** Benedict Cumberbatch, Martin Freeman  
**Creators:** Mark Gatiss, Steven Moffat

---

**Mark's rating**

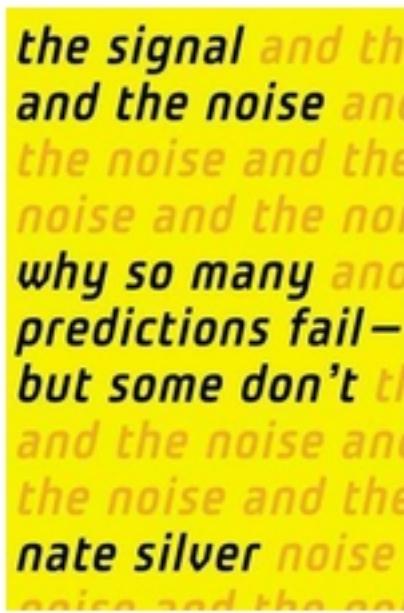
★★★★★

Not Interested

+ Instant Queue

*NetFlix*

# More Star Examples



Rate this book



really liked it

GoodReads

Introduction to Recommender Systems

T  
F  
by  
★  
N  
P  
b  
T  
O  
D  
—  
H  
P  
S  
E

Start here

1 How do you rate this item? ★★★★☆

2 Please enter a title for your review:

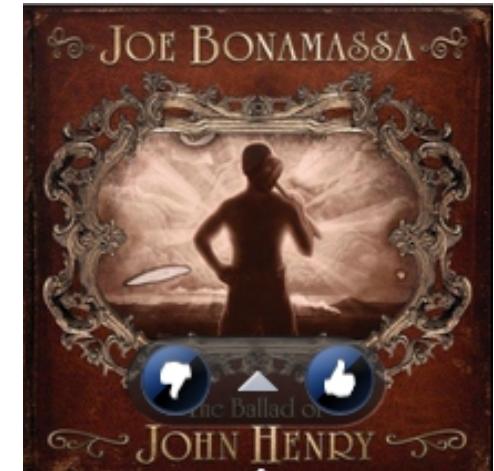
Amazon.com

★★★★★ = Must See  
★★★★☆ = Will Enjoy  
★★★☆☆ = It's OK  
★★☆☆☆ = Fairly Bad  
★☆☆☆☆ = Awful

MovieLens

# Thumbs and Likes

- Vote up/down
- Or just ‘Like’/‘+1’
- Common with ephemeral items
  - News aggregation (Reddit, Digg)
  - Q&A (StackOverflow)
  - YouTube
- Very low cost to rate



*Pandora*



*StackOverflow*



hot new rising controversial top wiki

want to join? login or register in seconds | English



Nextly Reddit app: Browse Reddit posts and comments side by side - try /r/funny! or your favorite sub-reddit ;) (nextly.com)

promoted by nextly  
8 comments share pocket



sponsored link what's this?

1 4881



[Grand Theft Auto V: Official Gameplay Video](#) (youtube.com)  
submitted 2 hours ago by SpaaceMILK to gaming

2883 comments share pocket

2 2982



[TIL The SEC was repeatedly and explicitly warned about the Madoff Ponzi scheme for 10 years by a forensic accountant. Over and over again they ignored the warnings and did nothing.](#) (npr.org)  
submitted 3 hours ago by Bluest\_waters to todayilearned

382 comments share pocket

3 6526



[Remember this guy?](#) (imgur.com)  
submitted 3 hours ago by saturn\_ to pics

1620 comments share pocket

4 2406



[Forever Torrenting.](#) (imgur.com)  
submitted 2 hours ago by thejarimteam to AdviceAnimals

246 comments share pocket

5 4126

↓



[James Bamford: "The NSA has no constitutional right to secretly obtain the telephone records of every American citizen on a daily basis, subject them to sophisticated data mining and store them forever. It's time government officials are charged with criminal conduct, including lying to Congress"](#) (blog.sfgate.com)  
submitted 5 hours ago by trot-trot to politics

1074 comments share pocket

6 2110

↓



[Hero Fukushima ex-manager who foiled nuclear disaster dies of cancer: It was Yoshida's own decision to disobey HQ orders to stop using seawater to cool the reactors. Instead he continued to do so and saved the active zones from overheating and exploding](#) (rt.com)  
submitted 3 hours ago by Carnival666 to worldnews

234 comments share pocket

7 2271

↓



[Am I doing it right?](#) (imgur.com)  
submitted 3 hours ago by mizzykid to funny

66 comments share pocket

8 1683

↓



[He's sat on this chair since he was a puppy. I guess he doesn't realize how big he's gotten...](#) (i.imgur.com)  
submitted 2 hours ago by wr3stler to aww

86 comments share pocket

9 1615

↓



[Official Grand Theft Auto V Gameplay video.](#) (youtube.com)  
submitted 2 hours ago by konvictkarl to videos

517 comments share pocket

search reddit

username password

 remember me [reset password](#)[login](#)[Submit a new link](#)[Submit a new text post](#)

# WHERE DO YOU STAND?

Take a stand for an  
Open Internet

[reddit.com/r/stand](#)[discuss this ad on reddit](#)

[reddit gold:](#) for that special someone.  
yes, you can give it to yourself.

# Other Interfaces

- Continuous scales
- Pairwise preference
- Hybrid (e.g. 1-100 + never again)
- Temporary (e.g. Pandora 30-day suspend)

# When are ratings provided?

- *Consumption* — during or immediately after experiencing the item
- *Memory* — some time after experience
- *Expectation* — the item has not been experienced

# Joke ratings

amazon Try Prime

Your Amazon.com | Today's Deals | Gift Cards | Sell | Help

Shop by Department ▾ Search Electronics ▾ Go

Hello, Sign in Your Account ▾ Try Prime ▾ Cart ▾ Wish List ▾

All Electronics Best Sellers Electronics Accessories Audio & Home Theater Camera & Photo Car Electronics & GPS Cell Phones & Accessories Computers MP3 Players TV & Video Trade-In

## Customer Reviews

### AudioQuest K2 Terminated Speaker Cable - UST 2.44 m Plugs 8' Pair

**232 Reviews**

|         |       |
|---------|-------|
| 5 star: | (128) |
| 4 star: | (40)  |
| 3 star: | (20)  |
| 2 star: | (12)  |
| 1 star: | (32)  |

Average Customer Review (232 customer reviews)

Share your thoughts with other customers

Create your own review

**Questions? Get fast answers from reviewers**

What do you want to know about this product?

See 1 question & its answers Ask

**The most helpful favorable review**

1,955 of 2,005 people found the following review helpful

**If only Heracles had such power!**  
If there is one cable I would whole-heartedly trust to my Chimera-hunting needs, this would be the cable. No other cable has the tensile strength to properly and efficiently garrote a lycanthrope, asphyxiate an Esquilax or even gag a mermaid. Last week, using my trusty AudioQuest K2 (retrofitted with lead weights, bright orange latex paint and a generous coating of...  
[Read the full review >](#)

Published on March 19, 2009 by Valannin

› See more **5 star, 4 star** reviews

**Vs.**

**The most helpful critical review**

7,816 of 7,923 people found the following review helpful

**I have only a little time...**  
We live underground. We speak with our hands. We wear the earplugs all our lives.  
  
PLEASE! You must listen! We cannot maintain the link for long... I will type as fast as I can.  
  
DO NOT USE THE CABLES!  
  
We were fools, fools to develop such a thing! Sound was never meant to be this clear, this pure, this... accurate. For a few short days, we...  
[Read the full review >](#)

Published on November 15, 2010 by Whisper

› See more **3 star, 2 star, 1 star** reviews

### This product

**AudioQuest K2 Terminated Speaker Cable - UST 2.44 m Plugs 8' Pair** by **Audioquest**  
Used & New from: **\$13,099.00**

Add to Wish List See buying options

### Search Customer Reviews

GO

Only search this product's reviews

#### Product Ads from External Websites

**AudioQuest DragonTail USB 2.0 Extender**  
\$16.79 + \$1.75 Est. Shipping  
Other World Computing

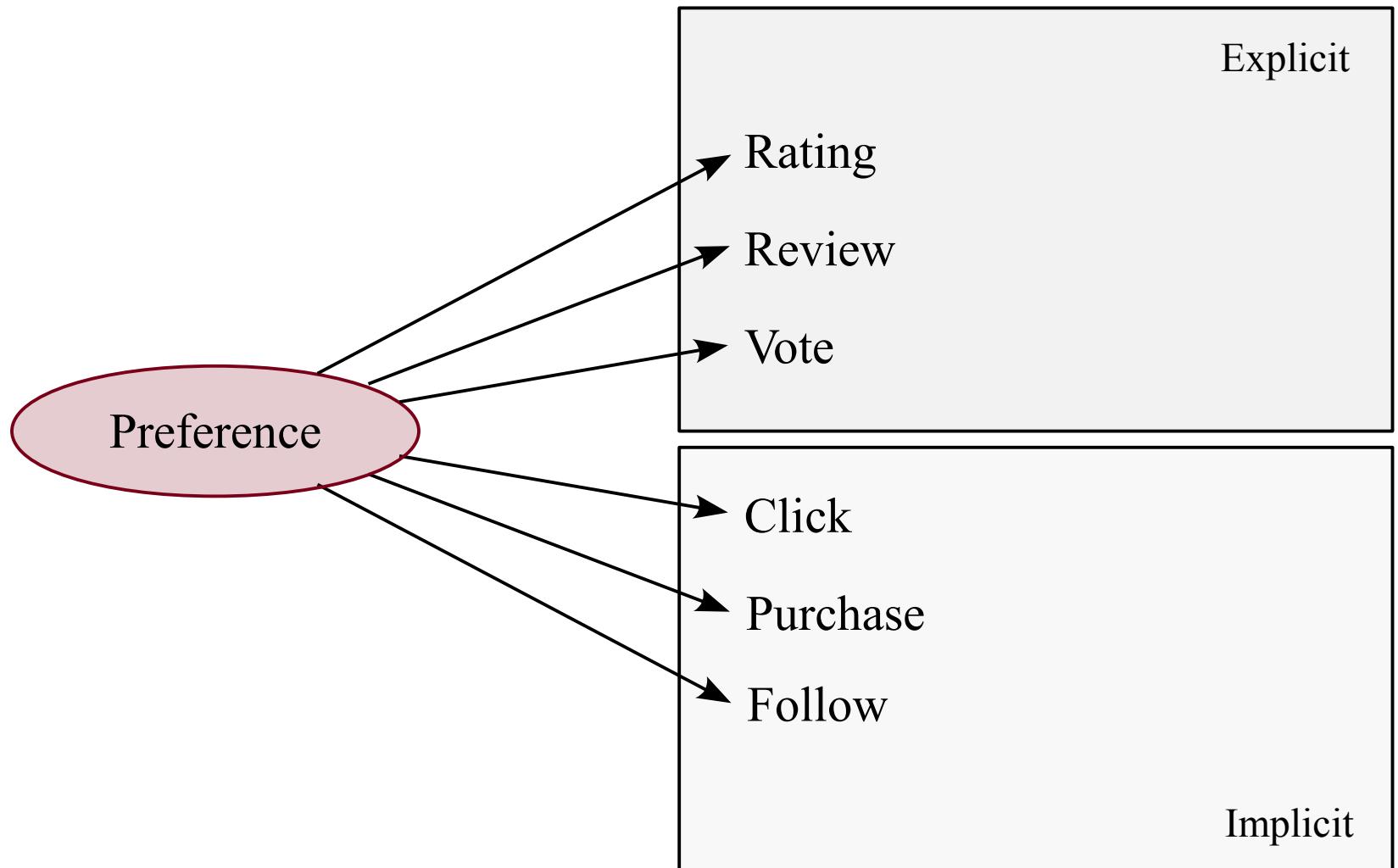
**110 Style 4 Pair Patch Plug / 5 Pack**  
\$15.19 + No Shipping Info  
CableOrganizers

Sponsored Content

# Difficulties with Ratings

- Are ratings reliable and accurate?
- Do user preferences change?
- What does a rating mean?

# Preference Model



# Implicit Data

- Data collected from user actions
- Key difference: user action is for some other purpose, not expressing preference
- Their actions say a lot!

# Reading Time

- Early implicit data: how long did user read?
- Listening and watching
  - IMMS
  - Video services

# Binary actions

- Click on link (ad, result, cross-reference)
- Don't click on link
- Purchase
- Follow/Friend

# Subtleties and Difficulties

- What does the action mean?
  - Purchase: they might still hate it
  - Don't click: expect bad, or didn't see
- How to scale/represent actions?
- Lots of opportunity to be creepy
  - Education may help
  - So can respecting privacy

# Conclusion

- Recommenders mine what users *say* and what they *do* to learn preferences
- Ratings provide explicit expressions of preference
- Implicit data benefits from greater volume

# 2-2: Preferences and Ratings

# 2-3: Predictions and Recommendations

# Learning Objectives

- To understand the ways in which recommender output can be used
- To understand the distinction between predictions and recommendations
- To understand the distinction between organic and explicit presentation
- To review examples and understand which presentation makes most sense in different applications.

# Predictions

- Estimates of how much you'll like an item
  - Often scaled to match some rating scale
  - Often tied to search or browsing for specific products



Publish your reviews on Google+ Local

Start Now

ADVERTISEMENT

## Hammacher Schlemmer

Offering the Best, the Only  
and the Unexpected Since 1848



SHOP NOW »



Welcome Joseph K | Sign Out | Help

# ZAGAT

(restaurant name, cuisine, feature ...)

(city, zip, street, neighborhood)

in

Minneapolis, MN

SEARCH



LISTS

VOTE

EVENTS

STORE

MOBILE

BLOG ↗

ZAGAT WINE ↗



## La Belle Vie

French (New), Mediterranean | Loring Park



Share Email

510 Groveland Ave. (Hennepin Ave.)  
Minneapolis, MN 55403  
[www.labellvie.us](http://www.labellvie.us)

612-874-6440

[Edit Info](#) | Are You the Owner?

## Make a Reservation

Date &amp; Time

06/29/2013

Party Size

7:00PM

2

[FIND A TABLE](#)

## ZAGAT RATINGS &amp; REVIEW

KEY TO RATINGS

FOOD

DECOR

SERVICE

COST

29

27

29

\$84

"Just close your eyes" and pick any of the "modern", "exquisite" Med-New French dishes (or go for the "brilliant" tasting menu) at this "wonderful little gem" in Loring Park, "top of the heap when it comes to fine dining", complete with a "formal look" in a "grande dame of an old apartment building", and earning the Twin Cities' No. 1 score for Service ("phenomenal"); sure it's a "splurge", so "bring your best credit card" and get set for a "memorable" evening; P.S. the "gorgeous" bar is "less stuffy" with a more affordable menu.

## STATS

# OF REVIEWS: 264

FRENCH (NEW) RANK: 1 of 2

LORING PARK RANK: 1 of 4

MEMBERS BEEN HERE: 15

MEMBER FAVORITES: 8

## ADD TO LISTS

Been Here

Favorites

Wish List

Custom Lists

OVERVIEW

REVIEWS

MENU

PHOTOS

PLAN YOUR VISIT

## MEMBER REVIEWS

[EDIT YOUR REVIEW](#)[See Reviews From Other Publications](#)

My wife and I treated ourselves to dinner at La Belle Vie for our wedding anniversary and it was a treat. We went with one of the tasting menus and it was a memorable meal!

Published June 21, 2012



**Steve E**  
164 Reviews  
Bloomington  
Member since:  
'Nov. 1999'

## GOOD TO KNOW

ATMOSPHERE Transporting Experience

FEATURES Game Served  
Private Room Available  
Winning Wine List

MEALS Dinner Served

SIGNATURE DISHES Beef Tenderloin  
Roasted Poussin

ADVERTISEMENT



**Bellagio**  
**\$159.00**  
Booking.com  
Best Price Guarantee



# The Camera Never Lies

Dr Emmett Sullivan

Film, images & historical interpretation in the 20th century for those who have a general interest in photojournalism, and films based on historical events.

**Workload:** 5-10 hours/week



Watch intro video

## Sessions:

Jun 24th 2013 (6 weeks long)

[Enroll for Free](#)

[Enroll in Signature Track](#)

i

Future sessions

[Add to Watchlist](#)

536

202

3.6k

Tweet

+1

Like

## About the Course

This short course is an introduction to use of photographs as historical evidence in the twentieth century, issues of authenticity and manipulation, and the place of film and historical adoption as public history.

## Course Syllabus

- Week 1: The Camera Never Lies - Introduction
- Week 2: Images and History in the Twentieth Century
- Week 3: The Air-Brushing of History: Stalin and Falsification
- Week 4: Photojournalism, Authenticity and Matters of Public Acceptability - The Battle of Mogadishu
- Week 5: The Power of the Image - Mount Suribachi, 1945
- Week 6: From Page to Screen - Film as Public History

## About the Instructor



Emmett Sullivan

University of London  
International Program...

# movielens

helping you find the *right* movies

Welcome konstan@cs.umn.edu ([Log Out](#))

You've rated 59 movies.

You're the 28th visitor in the past hour.

★★★★★ = Must See  
★★★★☆ = Will Enjoy  
★★★★☆ = It's OK  
★★☆☆☆ = Fairly Bad  
★★☆☆☆ = Awful

[Home](#) | [Find Movies](#) | [Q&A \(new\)](#) | [Preferences](#) | [Help](#)

## Shortcuts    Search

**Basic Search**

Title:

All Genres  All Dates

Domain:

Tag:

Use selected buddies!

Exclude your ratings

Exclude movies without predictions

## Select Buddies

- Dan C.  
 Yan Chen  
 John  
 herlocke@cs.umn.edu

[What are buddies?](#)

## Advanced Search

### Member Search

### Saved Searches

- [budy-new-comedy](#)
- [complex-hanks](#)
- [New Drama](#)

There are 45 movies matching your search:

Movies starring: **tom hanks**

You've sorted by: **Titles**

[Show Printer-Friendly Page](#) | [Download Results](#) | [Permalink](#) | [Suggest a Title](#)

Tags Related to Your Search: [Tom Hanks \(546\)](#), [Pixar \(151\)](#), [World War II \(85\)](#), [animation \(84\)](#), [true story \(74\)](#), [\(about tags\)](#)

Page 1 of 2

1 2 next

Skip to page #:

| Prediction or Rating ↗ | Your Rating                                      | Movie Information   | Wish List                |
|------------------------|--|---|--------------------------|
| ★★★                    | Not seen <input type="button" value="Not seen"/> | 'burbs, The (1989) DVD info imdb flag Movie Tuner<br>Comedy<br>[add tag] Popular tags: <a href="#">suburbia</a> <a href="#">thriller</a> <a href="#">Tom Hanks</a>  | <input type="checkbox"/> |
| ★★★★                   | Not seen <input type="button" value="Not seen"/> | Angels & Demons (2009) DVD info imdb flag Movie Tuner<br>Crime, Drama, Mystery, Thriller - Italian, Latin, English<br>[add tag] Popular tags: <a href="#">based on a book</a> <a href="#">action</a> <a href="#">Tom Hanks</a>  | <input type="checkbox"/> |
| ★★★★★                  | Not seen <input type="button" value="Not seen"/> | Apollo 13 (1995) DVD info imdb flag Movie Tuner<br>Adventure, Drama, IMAX<br>[add tag] Popular tags: <a href="#">based on a true story</a> <a href="#">based on a book</a> <a href="#">space</a>  | <input type="checkbox"/> |
| ★★★                    | Not seen <input type="button" value="Not seen"/> | Bachelor Party (1984) DVD info imdb flag Movie Tuner<br>Comedy<br>[add tag] Popular tags: <a href="#">Nudity (Topless)</a> <a href="#">Betamax</a> <a href="#">Neal Israel</a>  | <input type="checkbox"/> |
| ★★★★★                  | Not seen <input type="button" value="Not seen"/> | Band of Brothers (HBO) (2001) DVD info imdb flag Movie Tuner<br>Action, Adventure, Drama, War<br>[add/edit] Your tags: <a href="#">Tom Hanks, true story, based on a book</a><br>Popular tags: <a href="#">friendship</a> <a href="#">action</a> <a href="#">World War II</a> | <input type="checkbox"/> |
| ★★★★                   | Not seen <input type="button" value="Not seen"/> | Beyond All Boundaries (2009) info imdb flag Movie Tuner<br>Documentary, War<br>[add tag] Popular tags: <a href="#">4d</a> <a href="#">World War II</a> <a href="#">short film</a>   | <input type="checkbox"/> |
| ★★★★★                  | Not seen <input type="button" value="Not seen"/> | Big (1988) DVD VHS info imdb flag Movie Tuner<br>Comedy, Drama, Fantasy, Romance<br>[add tag] Popular tags: <a href="#">comedy</a> <a href="#">coming of age</a> <a href="#">Tom Hanks</a>  | <input type="checkbox"/> |
| ★★★                    | Not seen <input type="button" value="Not seen"/> | Bonfire of the Vanities (1990) DVD info imdb flag Movie Tuner<br>Comedy, Crime, Drama<br>[add tag] Popular tags: <a href="#">Morgan Freeman</a> <a href="#">Bruce Willis</a> <a href="#">Tom Hanks</a>  | <input type="checkbox"/> |
| ★★★★                   | Not seen <input type="button" value="Not seen"/> | Cast Away (2000) DVD VHS info imdb flag Movie Tuner<br>Drama  | <input type="checkbox"/> |

# Recommendations

- Recommendations are suggestions for items you might like (or might fit what you're doing)
  - Often presented in the form of “top-n lists”
  - Also sometimes just placed in front of you



konstan



konstan@umn.edu



+ Share



Web

Images

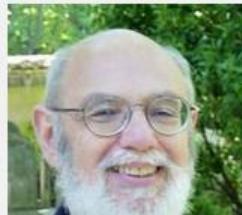
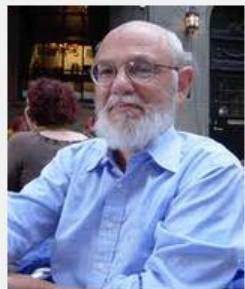
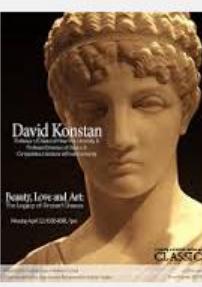
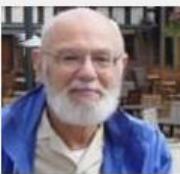
Maps

Shopping

More

Search tools

Safe Search



**Restrict** to movies found in  Any  All of the following selected Genres

- |                                      |                                    |  |
|--------------------------------------|------------------------------------|--|
| <input type="checkbox"/> Action      | <input type="checkbox"/> Drama     | <input checked="" type="checkbox"/> Sci-Fi |
| <input type="checkbox"/> Adventure   | <input type="checkbox"/> Fantasy   | <input type="checkbox"/> Thriller          |
| <input type="checkbox"/> Animation   | <input type="checkbox"/> Film-Noir | <input type="checkbox"/> War               |
| <input type="checkbox"/> Children    | <input type="checkbox"/> Horror    | <input type="checkbox"/> Western           |
| <input type="checkbox"/> Comedy      | <input type="checkbox"/> Musical   | <input type="checkbox"/> IMAX              |
| <input type="checkbox"/> Crime       | <input type="checkbox"/> Mystery   |  |
| <input type="checkbox"/> Documentary | <input type="checkbox"/> Romance   |  |

**Restrict** to movies NOT found in the following selected Genres

- |                                      |                                    |                                   |
|--------------------------------------|------------------------------------|-----------------------------------|
| <input type="checkbox"/> Action      | <input type="checkbox"/> Drama     | <input type="checkbox"/> Sci-Fi   |
| <input type="checkbox"/> Adventure   | <input type="checkbox"/> Fantasy   | <input type="checkbox"/> Thriller |
| <input type="checkbox"/> Animation   | <input type="checkbox"/> Film-Noir | <input type="checkbox"/> War      |
| <input type="checkbox"/> Children    | <input type="checkbox"/> Horror    | <input type="checkbox"/> Western  |
| <input type="checkbox"/> Comedy      | <input type="checkbox"/> Musical   | <input type="checkbox"/> IMAX     |
| <input type="checkbox"/> Crime       | <input type="checkbox"/> Mystery   |                                   |
| <input type="checkbox"/> Documentary | <input type="checkbox"/> Romance   |                                   |

**Restrict To Language:**

- |   |  |                                     |
|---|--|-------------------------------------|
| <input checked="" type="checkbox"/> English | <input type="checkbox"/> All Non-English |                                     |
| <input type="checkbox"/> Cantonese          | <input type="checkbox"/> Italian         | <input type="checkbox"/> Portuguese |
| <input type="checkbox"/> Dutch              | <input type="checkbox"/> Japanese        | <input type="checkbox"/> Russian    |
| <input type="checkbox"/> French             | <input type="checkbox"/> Korean          | <input type="checkbox"/> Silent     |
| <input type="checkbox"/> German             | <input type="checkbox"/> Mandarin        | <input type="checkbox"/> Spanish    |
| <input type="checkbox"/> Hindi              | <input type="checkbox"/> Polish          | <input type="checkbox"/> Swedish    |

Other:

**Restrict** to Movies with release Date (for whichever format is selected)

Within the Last 90 Days

In the range

From  To

**Restrict To:**  Your Ratings  Your HiddenList  Your WishList

**Restrict To Format:**  DVD  VHS

Don't Show Predictions!

Include Movies I've Already Rated!

**Sort By:**  Prediction or Rating  Prediction  Title  Number of Ratings  Release Date  Random  Date Added  Date of Rating Sort

Title Similarity

[How to use advanced search?](#)

# movielens

helping you find the *right* movies

Welcome konstan@cs.umn.edu ([Log Out](#))

You've rated 59 movies.

You're the 28th visitor in the past hour.

- ★★★★★ = Must See
- ★★★★☆ = Will Enjoy
- ★★★☆☆ = It's OK
- ★★☆☆☆ = Fairly Bad
- ★☆☆☆☆ = Awful

[Home](#) | [Find Movies](#) | [Q&A \(new\)](#) | [Preferences](#) | [Help](#)

## Shortcuts    Search

**Basic Search**

Title:

Sci-Fi  All Dates

Domain: All movies

Tag:

Use selected buddies!

Exclude your ratings

Exclude movies without predictions

**Select Buddies**

Dan C.  
 Yan Chen  
 John  
 herlocke@cs.umn.edu

[What are buddies?](#)

**Advanced Search**

**Member Search**

**Saved Searches**

- [budy-new-comedy](#)
- [complex-hanks](#)
- [New Drama](#)

There are 849 movies matching your search:  
 You've searched for movies with these languages: English  
 Movies with genres matching ALL of : Sci-Fi  
 Movies released in: 1970 -- 2013  
 Movies you've rated are Not Shown  
 You've sorted by: Prediction

[Show Printer-Friendly Page](#) | [Download Results](#) | [Permalink](#) | [Suggest a Title](#)

Tags Related to Your Search: [sci-fi](#) (2339), [dystopia](#) (1168), [aliens](#) (994), [time travel](#) (854), [action](#) (768), [\(about tags\)](#)

Page 1 of 34

1 2 3 4 ... 34 next

Skip to page #:

[Go](#)

| Prediction or Rating | Your Rating                               | Movie Information  | Wish List                |
|----------------------|---|--|--------------------------|
| ???                  | Not seen <input type="button" value="▼"/> | <a href="#">Minority Report (2002)</a> DVD info imdb flag Movie Tuner<br>Action, Crime, Mystery, Sci-Fi, Thriller<br><br>[add tag] Popular tags: <a href="#">post-apocalyptic</a>   <a href="#">sci-fi</a>   <a href="#">dystopia</a>            | <input type="checkbox"/> |
| ???                  | Not seen <input type="button" value="▼"/> | <a href="#">Spider-Man (2002)</a> DVD info imdb flag Movie Tuner<br>Action, Adventure, Sci-Fi, Thriller<br><br>[add tag] Popular tags: <a href="#">Action</a>   <a href="#">comic book</a>   <a href="#">superhero</a>                           | <input type="checkbox"/> |
| ???                  | Not seen <input type="button" value="▼"/> | <a href="#">Inception (2010)</a> DVD info imdb flag Movie Tuner<br>Action, Crime, Drama, IMAX, Mystery, Sci-Fi, Thriller<br><br>[add tag] Popular tags: <a href="#">alternate reality</a>   <a href="#">psychology</a>   <a href="#">surreal</a> | <input type="checkbox"/> |
| ???                  | Not seen <input type="button" value="▼"/> | <a href="#">V for Vendetta (2006)</a> DVD info imdb flag Movie Tuner<br>Action, IMAX, Sci-Fi, Thriller<br><br>[add tag] Popular tags: <a href="#">social commentary</a>   <a href="#">sci-fi</a>   <a href="#">dystopia</a>                      | <input type="checkbox"/> |
| ???                  | Not seen <input type="button" value="▼"/> | <a href="#">Spider-Man 2 (2004)</a> DVD info imdb flag Movie Tuner<br>Action, Adventure, IMAX, Sci-Fi<br><br>[add tag] Popular tags: <a href="#">action</a>   <a href="#">comic book</a>   <a href="#">superhero</a>                             | <input type="checkbox"/> |
| ???                  | Not seen <input type="button" value="▼"/> | <a href="#">Prestige, The (2006)</a> DVD info imdb flag Movie Tuner<br>Drama, Mystery, Sci-Fi, Thriller<br><br>[add tag] Popular tags: <a href="#">based on a book</a>   <a href="#">twist ending</a>   <a href="#">nonlinear</a>                | <input type="checkbox"/> |
| ???                  | Not seen <input type="button" value="▼"/> | <a href="#">Iron Man (2008)</a> DVD info imdb flag Movie Tuner<br>Action, Adventure, Sci-Fi - English, Persian<br><br>[add tag] Popular tags: <a href="#">comic book</a>   <a href="#">sci-fi</a>   <a href="#">superhero</a>                    | <input type="checkbox"/> |
| ???                  | Not seen <input type="button" value="▼"/> | <a href="#">Star Trek (2009)</a> DVD info imdb flag Movie Tuner<br>Action, Adventure, IMAX, Sci-Fi<br><br>[add tag] Popular tags: <a href="#">time travel</a>   <a href="#">sci-fi</a>   <a href="#">space</a>                                   | <input type="checkbox"/> |
| ???                  | Not seen <input type="button" value="▼"/> | <a href="#">X-Men (2000)</a> DVD info imdb flag Movie Tuner<br>Action, Adventure, Sci-Fi, Superhero<br><br>[add tag] Popular tags: <a href="#">superhero</a>   <a href="#">sci-fi</a>   <a href="#">x-men</a>                                    | <input type="checkbox"/> |



## Top Selling Phone &amp; Accessories

## Top Selling Smartphones

Samsung Galaxy  
Note II White 3GSamsung Galaxy  
Note II Gray 3GBlackBerry Z10  
16GB Black 3GSamsung Galaxy S3  
16GB White 3GSony Xperia ZL  
Black 4G LTESamsung Galaxy S3  
16GB Blue 3GSony Xperia ZL  
Black 3GSamsung Galaxy S3  
16GB BlackLG Nexus 4 Black  
3GSony Xperia Z Black  
4G

## Top Unlocked Budget Phones

Sony Xperia Tipo  
Black 3G

HTC Desire A Brown

Motorola XT532  
SilverLG Optimus L3  
Black

Nokia 100 Black

Blu Samba JR Light  
BlueLG Rumour Plus  
Blue 3GSamsung Eternity II  
Blue 3GBlu Jenny  
Black/Blue

Pantech Link Wine

## Newest Unlocked Phones

Samsung Galaxy S4  
16GB White 3GSamsung Galaxy S4  
16GB Black 3GSony Xperia ZL  
16GB Black 4G LTESony Xperia ZL  
16GB Black 3GSony Xperia Z 16GB  
Black 4GBlackBerry Z10  
16GB Black 3GBlackBerry Z10  
16GB Black 3G LTELG Nexus 4 16GB  
Black 3GLG Nexus 4 8GB  
Black 3GNokia Lumia Black  
3G New software Keeping up with  
friends and family Special occasions Price drops

Vote ►

Results ►

**EARN  
50,000**  
Membership Rewards®  
points

after you spend \$5,000  
in purchases on the Card  
in the first 3 months  
of Card membership.  
Apply online for this offer.†

LEARN MORE

+Terms &amp; Limitations Apply



The Business Gold Rewards Card

OPEN

## Ads by Google (2)

[Cheap Unlocked](#)[Phones@\\$37](#)

Latest Unlocked

Quadband Cellphones

From \$37. Huge

Range+Free Shipping

[www.everbuying.com/Unlocked\\_Phones](#)[Unlocked Android Phone](#)

Bestselling Unlocked

Android Phones Low

Price, Fast Shipping. Buy

# Prediction and Recommendation

- Often, the two come together
- Predictions:
  - Pro: helps quantify item
  - Con: provides something falsifiable
- Recommendations
  - Pro: provides good choices as a default
  - Con: if perceived as top-n, can result in failure to explore (if top few seem poor)



amazon.com

Click here to enable desktop notifications for University of Minnesota Mail. [Learn more](#) [Hide](#)

Mail



konstan@umn.edu



+ Share



9 of many



COMPOSE

Inbox

Sent Mail

Drafts (23)

Spam (514)

#todo

Academic-Admin

SocComp

CHI 2012

E-mail

Keynote

To Do

Classes

CSci 1001

CSci 5115

Project-Cont...

CSci 5116

CSci 5125 (399)

CSci 5129

CSci 8001 (5)

CSci 8115 (2)

CSci5115

e-Pub-Health

PhD

recsys

SEng5115-S04

SW5115-F00

Tutorials



Search people...

D Singer

Ed H. Chi

Ellen Konstan

Paul Resnick

Joseph A. Konstan,

Are you looking for something in our Ice Makers department? If so, you might be interested in these items.

### Ice Makers

[NewAir AI-100R 28-Pound Portable Icemaker, Red](#)  
by NewAir[Learn more](#)[Add to Wish List](#)List Price: \$271.36  
Price: \$187.98   
You Save: \$83.38 (31%)[Portable Stainless Steel Ice Maker With Digital Display](#)  
by EdgeStar[Learn more](#)[Add to Wish List](#)Price: \$140.05  
Ships from and sold by [CompactAppliance](#).[Polar PIM10BLS Stainless Steel Portable Ice Maker](#)  
by Greenway Home Products[Learn more](#)[Add to Wish List](#)List Price: \$179.99  
Price: \$177.27   
You Save: \$2.72 (2%)[Sunpentown IM-101S Portable Ice Maker with LCD with](#)  
Stainless Steel...  
by Sunpentown[Learn more](#)[Add to Wish List](#)List Price: \$349.99  
Price: \$235.00   
You Save: \$114.99 (33%)

Related Google+ Page

Amazon.com

[Follow](#)

Order before 1pm PST for **FREE Next Business Day shipping** on all **Clothing**. [Learn More](#)

24/7 Customer Service (800) 927-7671

Help

Live Help

Log In or Register

My Account

My Favorites



Shoes, Clothing, Bags, etc.

SEARCH

CLEARANCE

SHOP NOW

**FREE SHIPPING  
AND  
FREE RETURNS**  
365-DAYS A YEAR



MY CART

SEARCH BY: [Size](#), [Narrow Shoes](#), [Wide Shoes](#), [Popular Searches](#)

SHOES

CLOTHING

BAGS &amp; HANDBAGS

AT HOME

BEAUTY

ACCESSORIES

SHOP BY...

WOMEN'S

MEN'S

KIDS'

ALL DEPARTMENTS ▾

ALPHABETICAL BRAND INDEX

# · A · B · C · D · E · F · G · H · I · J · K · L · M · N · O · P · Q · R · S · T · U · V · W · X · Y · Z

**"Men Shoes"** we found **360 items!**Sort By [Relevance](#)

NARROW YOUR CHOICES

YOUR SELECTIONS: ATHLETIC ✕ SHOES ✕ MEN ✕ 10 ✕ EE ✕

**Shop All  
SALE ITEMS**

[SHOW SALE ITEMS](#)

MEN'S SIZE

|      |      |     |      |      |
|------|------|-----|------|------|
| 3.5  | 4    | 4.5 | 5    | 5.5  |
| 6    | 6.5  | 7   | 7.5  | 8    |
| 8.5  | 9    | 9.5 | 10   | 10.5 |
| 11   | 11.5 | 12  | 12.5 | 13   |
| 13.5 | 14   | 15  | 16   | 17   |
| 18   |      |     |      |      |

MEN'S WIDTH

|    |    |   |    |    |
|----|----|---|----|----|
| N  | M  | W | WW | 2A |
| B  | D  | E | EE | 3E |
| 4E | 6E |   |    |    |

CATEGORY

Sneakers & Athletic Shoes (223)  
Boots (127)  
Sandals (10)  
Clogs & Mules (2)

NEW



**ASICS**  
GEL-Nimbus® 15  
\$145.00

NEW



**New Balance**  
MX797v2  
\$84.95

NEW



**New Balance**  
MX797v2  
\$84.95

NEW



**New Balance**  
MX797v2  
\$84.95

SALE



**ASICS**  
GEL-Nimbus® 14  
\$124.95

SALE



**ASICS**  
GT-1000™  
\$89.95



**Nike**  
Air Pegasus+ 29  
\$89.95-\$100.00



**Brooks**  
Adrenaline™ GTS 13  
\$110.00

# Another dimension to consider

- How explicit is the prediction or recommendation (vs. organic)?
  - Historical note: we paid for it, we'll let you know
  - Today: balance between explicit prediction (falsifiable) and coarser granularity (you might like this!)
  - Today: balance between theses are the best (top-n) and softer presentation (here are some that might be interesting)

# You should now understand

- Difference between prediction and recommendation
- Range of explicit to organic for both predictions and recommendations
- Advantages and disadvantages in both dimensions.

# 2-3: Predictions and Recommendations

# 2-4: Scoring and Ranking

# Introduction

- Last 2 lectures:
  - how to collect data
  - what we present to users
- This lecture: how to do it
  - what predictions to show
  - how to rank

# Learning Objectives

- Understand several ways of computing and displaying predictions
- Understand how to rank items with sparse, time-shifting data
- Understand several points in the design space for prediction and recommendation, and some of their tradeoffs

# Overview

- Example
- Displaying Aggregate Preferences (*predict*)
- Ranking Items (*recommend*)

# Overview

- Example
- Displaying Aggregate Preferences (*predict*)
- Ranking Items (*recommend*)

# Example - Reddit

- Social news aggregator
- Non-personalized news recommender
- Users vote on items to determine top item



hot new rising controversial top wiki

want to join? login or register in seconds | English



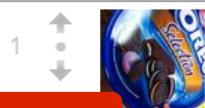
You can't change the world without crashing the system. Watch the new trailer for The Fifth Estate and see the movie in theaters Oct 18. ([www.youtube.com](http://www.youtube.com))

promoted by redditads

160 comments share pocket



sponsored link what's this?



To the user who wanted all the oreos in one box. ([imgur.com](http://imgur.com))

submitted 1 hour ago by EpicRay to pics

309 comments share pocket

search reddit

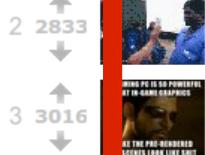
username password

 remember me [reset password](#)

login

Submit a new link

Submit a new text post



It's magic! ([i.imgur.com](http://i.imgur.com))

submitted 3 hours ago by MrBurd to funny

149 comments share pocket



First world gaming problem after getting a high-end gaming PC and Deus Ex ([i.imgur.com](http://i.imgur.com))

submitted 3 hours ago by foehammer111 to gaming

678 comments share pocket



The British government has invested £60m to develop the potentially revolutionary SABRE engine, designed to allow spacecraft to launch from a runway directly into low earth orbit. If the concept works, it could cut the cost of reaching LEO by 95%. Test flights planned for 2019. ([guardian.co.uk](http://guardian.co.uk))

submitted 4 hours ago by JB\_UK to technology

1 comments share pocket



5 1917 NSA spying under fire | In a heated confrontation over domestic spying, members of Congress said Wednesday they never intended to allow the National Security Agency to build a database of every phone call in America. And they threatened to curtail the government's surveillance authority.

([news.yahoo.com](http://news.yahoo.com))

submitted 2 hours ago by douglasmacarthur to news

207 comments share pocket

 search reddit  
 username  password  
 remember me [reset password](#) 

Submit a new link

Submit a new text post



discuss this ad on reddit



6 2155 1999 Newspaper Article says AVATAR Would Never be made into a Movie

([static.squarespace.com](http://static.squarespace.com))

submitted 4 hours ago by solidfox535 to movies

393 comments share pocket

This year, give the gift of [reddit gold](#).  
(and you should probably also give some other, better gifts)



7 2489 Couldn't stop myself... ([livememe.com](http://livememe.com))

submitted 4 hours ago by crazythoughts to AdviceAnimals

120 comments share pocket



8 1600 Two years after the nuclear disaster, here's what the plantlife in Fukushima Prefecture looks like. ([Inaccurate](#)) ([imgur.com](http://imgur.com))

submitted 2 hours ago by caadbury to WTF

469 comments share pocket



9 3245 Norwegian Bachelor Party, Groom fooled into thinking that he is going bungee jumping from an old bridge ([youtube.com](http://youtube.com))

submitted 7 hours ago by Wish to videos

835 comments share pocket



10 2146 Spanish scientists successfully generate "artificial bones" from umbilical cord stem cells ([canal.ugr.es](http://canal.ugr.es))

submitted 5 hours ago by Libertaea to science

...yahoo.com/nsa-spying-under-fire-youve-got-proble...



# Overview

- Example
- **Displaying Aggregate Preferences (*predict*)**
- Ranking Items (*recommend*)

# Simple Display Approaches

- Average rating / upvote proportion
- Net upvotes / # of likes
- $\% \geq 4$  stars ('positive')
- Full distribution

# Goal of Display

*To help users decide to buy/read/view the item.*

# Simple Display Approaches (again)

- Average rating / upvote proportion
  - Of people who vote, do they like it?
  - Doesn't show popularity
- Net upvotes / # of likes
  - Shows popularity
  - No controversy
- $\% \geq 4$  stars ('positive')
- Full distribution

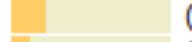
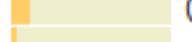
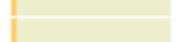
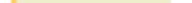
# Example: Amazon.com

Canon PowerShot A2300 16.0 MP Digital Camera with 5x Di  
Wide-Angle Lens with 720p HD Video Recording (Red)

by [Canon](#)

 (135 customer reviews)

**4.2 out of 5 stars**

|                         |   |      |
|-------------------------|---|------|
| <a href="#">5 star:</a> |  | (78) |
| <a href="#">4 star:</a> |  | (29) |
| <a href="#">3 star:</a> |  | (17) |
| <a href="#">2 star:</a> |  | (5)  |
| <a href="#">1 star:</a> |  | (6)  |

[See all 135 reviews](#)

“ It is easy to use and takes great pictures and video. ”

Marc Jaffrey | 39 reviewers made a similar statement

“ I highly recommend it for other point-and-shooters as well. ”

Troldilocks | 12 reviewers made a similar statement

“ It easily fits in my purse or pocket for easy carrying. ”

Lois | 14 reviewers made a similar statement

# Reddit

- 4 ↑ 3021 The British government has invested £60m to develop the potentially revolutionary SABRE engine, designed to allow spacecraft to launch from a runway directly into low earth orbit. If the concept works, it could cut the cost of reaching LEO by 95%. Test flights planned for 2019. (guardian.co.uk)  
submitted 4 hours ago by JB\_UK to technology  
645 comments share pocket
- 5 ↑ 1917 NSA spying under fire | In a heated confrontation over domestic spying, members of Congress said Wednesday they never intended to allow the National Security Agency to build a database of every phone call in America. And they threatened to curtail the government's surveillance authority.  
(news.yahoo.com)  
submitted 2 hours ago by douglasmacarthur to news  
207 comments share pocket
- 6 ↑ 2155 ↓ (NOT) COMING TO A THEATER NEAR YOU 1999 Newspaper Article says AVATAR Would Never be made into a Movie  
(static.squarespace.com)  
submitted 4 hours ago by solidfox535 to movies  
393 comments share pocket

# Overview

- Example
- Displaying Aggregate Preferences (*predict*)
- **Ranking Items (*recommend*)**

# Ranking

- What do you put at the top of Reddit?
- What is at the top of the e-Bay search list?
- You don't have to rank by prediction

# Why not rank by score?

- Too little data (one 5-star rating)
- Score may be multivariate (histogram)
- Domain or business considerations
  - Item is old
  - Item is ‘unfavored’

# Ranking Considerations

- Confidence
  - How confident are we that this item is good?
- Risk tolerance
  - High-risk, high-reward
  - Conservative recommendation
- Domain and business considerations
  - Age
  - System goals

# Damped means

- Problem: low confidence w/ few ratings
- Solution: assume that, without evidence, everything is average
- Ratings are evidence of non-averageness
- $k$  controls strength of evidence required

$$\frac{\sum_u r_{ui} + k\mu}{n + k}$$

# Confidence Intervals

- From the reading: lower bound of statistical confidence interval (95%)
- Choice of bound affects risk/confidence
  - Lower bound is conservative: be sure it's good
  - Upper bound is risky: there's a chance of amazing
- Reddit uses Wilson interval (for binomial) to rank comments

# Domain Consideration: Time

- Reddit: old stories aren't interesting
  - even if they have many upvotes!
- eBay: items have short lifetimes

# Scoring news stories

- Hacker News

$$\frac{(U - D - 1)^\alpha}{(t_{\text{now}} - t_{\text{post}})^\gamma} \times P$$

- Net upvotes, polynomially decayed by age
- Old items scored mostly by vote
- Multiplied by item penalty terms
  - incorporate community goals into score

# Reddit algorithm (c. 2010)

$$\log_{10} \max(1, |U - D|) + \frac{\text{sign}(U - D) t_{\text{post}}}{45000}$$

- Log term applied to votes
  - decrease marginal value of later votes
- Time is seconds since Reddit epoch
- Buries items with negative votes
- Time vs. vote impact independent of age
- Scores news items, not comments

# Ranking Wrap-Up

- There are some theoretically grounded approaches (confidence interval, damping)
- Many sites use ad-hoc methods
- Most formulas have constants, will be highly service-dependent
- Can manipulate for ‘good’ or ‘evil’
- Build based on domain properties, goals

# Predict with sophisticated score?

- Theoretically a fine thing to do
- Be careful with transparency/scrutability
  - If you say ‘average rating’ for damped mean, and show ratings, users may be confused
  - Most important case (low ratings) also easiest to hand-verify

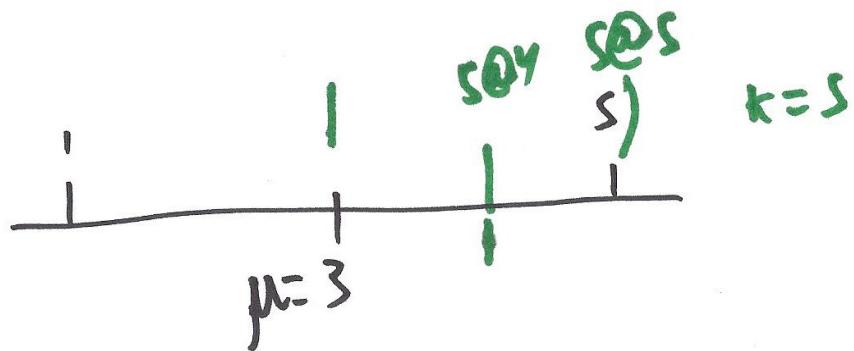
# Conclusion

- Sparsity, inconsistency, temporal concerns make data messy
- Simple scoring doesn't necessarily match the domain or business
- There are good ways to deal with this (decay, time, penalties, damping)
- We'll see more normalizations later

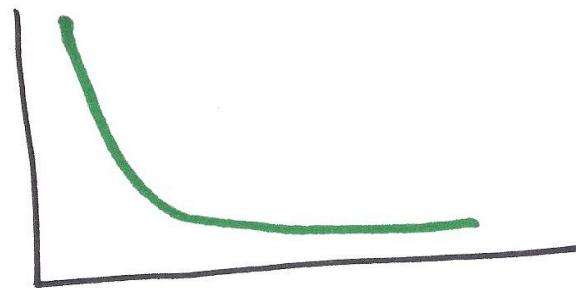
# 2-4: Scoring, Ranking, and Normalization

$$\sum_i r_{ui} + k\mu$$

$\cap \stackrel{k}{\Sigma}$  strength  
required



$$\frac{\frac{\text{Net upvotes}}{(U - D - 1)^{\alpha=0.8}}}{\frac{(t_{\text{now}} - t_{\text{post}})^{r=1.9} \cdot P}{\text{Lage}}}$$



$$\log_{10} \max(1, |u-d|) + \frac{\text{sign}(u-d) t_{\text{post}}}{45,000}$$

Pred.

$$P_{a,i} = \frac{\sum_{v=1}^n r_{v,i}}{n}$$

↑↑  
User item  
a i

ratings

$$P_{a,i} = \bar{r}_v + \frac{\sum_{v=1}^n (r_{v,i} - \bar{r}_v)}{n}$$

$$P_{a,i} = \frac{\sum_{v=1}^n r_{v,i} \cdot w_{a,v}}{\sum_{v=1}^n w_{a,v}}$$

$$P_{a,i} = \bar{r}_a + \frac{\sum_{v=1}^n (r_{v,i} - \bar{r}_v) * w_{a,v}}{\sum_{v=1}^n w_{a,v}}$$

similarity  
predictive-  
ness

$$w_{a,v} = \frac{\sum_{i=1}^m (r_{a,i} - \bar{r}_a)(r_{v,i} - \bar{r}_v)}{\sigma_a \sigma_v}$$

- m small?
- unary?

top-n?

# 3-1: Introduction to Content-Based Recommenders

# Learning Objectives

- To understand the range and value of content-based approaches to recommendation
  - Pure information filtering systems
  - Case-based reasoning systems
  - Knowledge-based navigation systems
- To understand the strengths and drawbacks of content-based recommender systems

# Basic Idea: Stable Preferences

- Let's consider some examples:
  - News – I prefer stories on technology, University of Minnesota, Minnesota Vikings, restaurant reviews
  - Clothing – I prefer cotton, blue, low-priced, casual
  - Movies – I prefer Tom Hanks, Sandra Bullock, Woody Allen, Comedy
  - Hotels – I prefer 24-hour front desk, room service, internet, pool

# The key ideas

- Model items according to relevant attributes
- Model or reveal user preferences by attribute
- Voila! A Recommender

# Content-Based Filtering

- Key concept: building a vector of attribute or keyword preferences
- Example: Krakatoa Chronicle

<http://www.w3.org/Conferences/WWW4/Papers/93>

Kamba, Bharat, and Albers (WWW '95)

Document URL: <http://homepark.cc.gatech.edu:8080/cgi-bin/oldHJnews.pl?Read=Read+Last+Edition+of+Krakatoa+Chronicle>

Tuesday, April 26, 1995

1/20

# Krakatoa Chronicle



density

dense

tendency

general

sensitivity

not very sensitive

P: Peek an article, S: Save an article to a scrapbook

**P** **S** Technical static fuzzes Europe's telecom networks

(c) 1995 The News &amp; Observer Publishing Co.

(c) 1995 Bloomberg Business News

PARIS, April 26 (04-25-95) — With today's modems and telephone networks, sending data via telephone from a conference in Germany to your London office should be child's play, right?

Wrong.

Electric plugs differ from place to place. So do telephone connectors,

Interesting

dial tones and the latest transmission standards. As business travelers can attest, a thicket of technical, regulatory and other hurdles must be cleared before Europe becomes a "plug-and-play" \$160 billion telecommunications market, as promised, by 1998.

It will have to be in place before doctors in Belgium and Germany

**P** **S** President will buy Wang stake and invest in a Shanghai plant

(c) 1995 The News &amp; Observer Publishing Co.

(c) 1995 Bloomberg Business News

TAIPEI, April 26 (04-25-95) — President Technology Inc., a Taiwan computer maker and

software services concern, said it purchased the 49% of a Taiwan software concern it didn't already own from Lowell, Mass.-based Wang Laboratories Inc for NT\$365

million (\$14.4 million).

The company, 19% owned by Taiwan food processor President Enterprises Corp. (1216 TT), also said it plans to invest NT\$100 mil-

No Comment

Rogers, who is president and chief executive of the cable-television operator, was Unitel's chairman, a post he had held since mid-January. Lind and Gergacz served as vice chairmen of Unitel.

Last week, Rogers Communications said it wouldn't exercise an option to buy a majority stake in Unitel. Toronto-based Rogers acquired the option in January from

**P** R.R. Donnelley unit completes merger with Corporate Software

Interesting

(c) 1995 The News &amp; Observer Publishing Co.

(c) 1995 Bloomberg Business News

Chicago, April 24 (04-24-95) —

R.R. Donnelley & Sons said it has completed the merger of its

No Comment

lion in a computer monitor plant in Shanghai this year and double its capital to NT\$1.5 billion.

President Technology purchased

**P** Creative Technology, Aztech expand share of sound card market

(c) 1995 The News &amp; Observer Publishing Co.

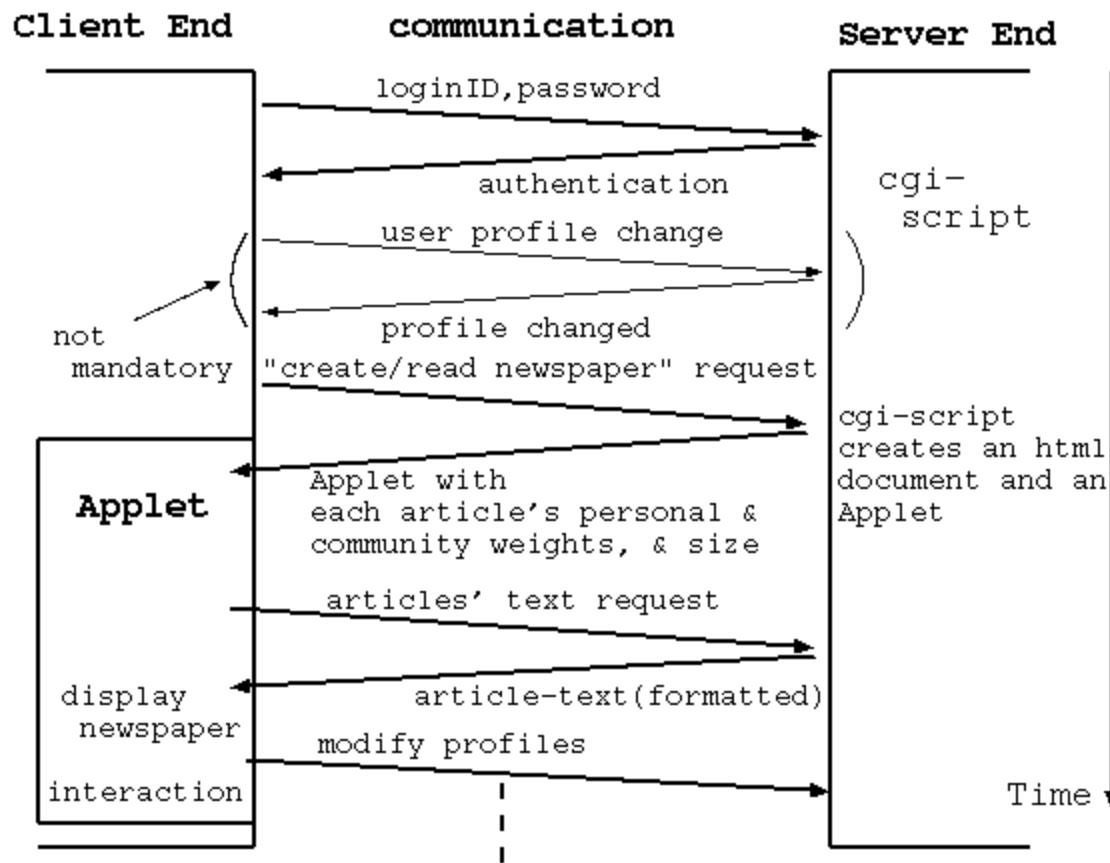
(c) 1995 Bloomberg Business News

Singapore, April 25 (04-24-95) —

Interesting

Creative Technology Ltd. and Aztech Systems Ltd. both expanded





# Wide range of Possibilities

- User could build own profile (awkward)
  - But allowing user to edit a profile can be valuable
- Infer profile from user actions
  - Read, Buy, Click
- Infer profile from explicit user ratings
  - How to map from item preference to attribute preference
- We merge actions/explicit into infer from ratings (explicit and implicit)

# How to build preferences?

- Let's start with the idea of a set of “keywords” that users may like, dislike, or not have an opinion on
- We could simply count the number of times the user chooses (or fails to choose) items with each keyword
- Or we can get more sophisticated
- More to come (future lecture) ...

# How to use preferences

- Given a vector of keyword preferences
  - Do we just add up likes and dislikes?
  - Can we figure out which keywords are more and less relevant?
- Forward reference: TFIDF

# Content-Based Recommenders

- Our assignments will be based on this model
  - Hand exercises: building a profile and using it to predict a few cases
  - Programming exercise: building a content-based recommender
- But first, a few other approaches ...

# Case-Based Recommendation

- The concept:
- Structure a database of cases around a set of relevant attributes (e.g., camera price, zoom, pixels)
- Query based on an example or attribute query, and retrieve relevant cases
- Open issue: Many ways to structure interaction

# etown's Ask Ida

- No longer exists (old screenshots)
- Uses an interview process to elicit preferences over attributes
- Uses preferences to recommend products
- Uses recommendation as a point to elicit further preferences
- Note: not intended as permanent preferences – just transactional



etown.com™

Customer Service | Your Account | Shopping Cart

July 17, 2000

## THE Consumer Electronics Source

Expert Advice, Comparison Shopping, Online Purchasing

[Should I buy a digital \(DV format\) camcorder?](#)

click here

Search

Products

Search

Advanced Search

## News & Views

### This Just In

- [Dynamo shoes generate electricity](#)
- [Showtime phases in Dolby 5.1 sound](#)
- [EMI to vend pay-per-download songs](#)
- [Streaming A/V on your cell phone?](#)
- [Stephen King plans serial e-book](#)
- [Juno, Hughes team on Web-by-satellite](#)
- [Will Sony Palm unit hit U.S. by Xmas?](#)

[more...](#)

### New Reviews/Products

- [EXCLUSIVE: Toshiba 480p-out DVD](#)
- [Bush TV center has nice price](#)
- [Vidikron projector: 'an eye-opener!'](#)
- [Panamax nixes electrical spikes](#)
- [JVC boombox has stylish looks](#)
- [DSS security recommends AT&T phone](#)
- [EXCLUSIVE: Sharp DVD player](#)
- [EXCLUSIVE: Philips/TiVo recorder](#)
- [EXCLUSIVE: Outlaw 6.1 A/V receiver](#)

[more...](#)

### Features / Columns

- [Unclear on the THX concept?](#)
- [In the Mix: Did we ask for DVD-A?](#)
- [Don't miss our updated FAQ for DTV!](#)
- [Camcorder Corner: Fade to black](#)
- [How to set up your subwoofer](#)

[INTERACTIVE](#)

## Browse & Buy

### Home Theater

- VCR Satellite DVD Receivers
- Surround Separates Speakers
- Packaged Systems Remotes
- Accessories PVR

### Television

- Direct-View TV Rear-Projection TV
- Front-Projection TV Flat-Panel TV
- DTV/HDTV Personal TV TV/VCR

### Camcorders

- 8mm Formats VHS Formats
- DV Format Digital Cameras

### Home Audio

- CD Players Tape/Disc Recorders
- Speakers Compact Systems Phono
- Integrated Amps Stereo Separates
- Accessories Multimedia Speakers
- A/V Furniture

### Portable Tech

- Boomboxes Personal Cassette
- Personal CD MiniDisc Palm PCs
- Portable Radios Digital Cameras
- MP3 Players Headphones
- 2-way Radios GPS

### Telecom

- Cordless Phones Cell Phones
- Phone/Answering Combos Pagers
- Corded Telephones Fax Machines
- Answering Machines

### Accessories

- Remote Controls Audio Cables
- Video Cables A/V Furniture Antennas
- Cases/Bags Surge Suppressors

### top 10 best sellers



Name brand products  
at discount prices

Great savings on  
every item!

**Cambridge SoundWorks model 188 Table-Top Radio**



List Price \$249.99  
**Sale \$149.99**

**Sharp MDMS722 Minidisc Portable Recorder**



List Price \$299.99  
**Free Shipping!  
\$169.70**

**JVC GR-DVM50**



High Quality Low Price  
**\$879.88**

**JBL PSWD 112**

List Price \$1499.95  
**Free Shipping!  
\$879.88**

**Dealers**

The Top 10 best-selling items:

File Edit View Favorites Tools Help

Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss

Address: http://www.etown.com/sales\_bot/index.jhtml;sessionid\$NV4NPYAAACBBJUPZJEHSFEQ Go Links >

etown.com™

Customer Service | Your Account | Shopping Cart July 17, 2000

**etown.com** consumer electronics **Summer Sale** Save Over 70% Click Here

**Categories**

- Home Theater
- Camcorders
- Home Audio
- Television
- Portable Tech
- Telecom

**News / Views**

- This Just In
- Reviews/Products
- Features/Columns
- Community Hall
- Knowledge Bank
- About etown.com
- CE Links

**NEW EPISODE!** What is GPS?

Interactive Knowledge Bank GET FLASH

**Ida: Your Interactive Decision Assistant**

**Welcome!**  
I'm Ida, your interactive decision advisor. I can help you find the products that best suit your needs and preferences. I combine **etown.com**'s expertise on consumer electronics with state-of-the-art artificial-intelligence techniques from **Ask Jeeves**. I think you'll find shopping for electronics can be easy and fun!

Select the product category you are interested in:

|  |   |
|--|---|
| Home Theater   | Portable Tech   |
| <ul style="list-style-type: none"><li>DVD Players</li><li>A/V Receivers</li><li>VCRs</li></ul> | <ul style="list-style-type: none"><li>Digital cameras</li><li>Boombboxes</li><li>Handheld/Palm PC</li></ul> |
| Home Audio   | Telecom   |
| <ul style="list-style-type: none"><li>Compact Systems</li><li>CD Players</li></ul>             | <ul style="list-style-type: none"><li>Cordless Phones</li></ul>   |
| Camcorders   |   |
|  | <ul style="list-style-type: none"><li>8mm, VHS, and DV</li></ul>  |

**Search**

Products  Advanced Search

QuickJump

Need expert advice? Ask Ida!

LIVE HELP

CLICK HERE to chat with our customer service

Customer Service 24 hours a day 7 days a week

Toll Free 877-GO-ETOWN

CHAT! with our experts

Audio Talk Played LOUD! with Michael Fremer 8pm Tuesdays

Start Quick Y:\book Wine.c... Y:\book xterm Shortc... Cannot... G:\pap... emacs... Calend... etow... 713tod... Micros... Micros... Internet 1:37 PM EN X



Customer Service | Your Account | Shopping Cart

July 17, 2000

## Change the way you think...

### Categories

- [Home Theater](#)
- [Camcorders](#)
- [Home Audio](#)
- [Television](#)
- [Portable Tech](#)
- [Telecom](#)

### News / Views

- [This Just In](#)
- [Reviews/Products](#)
- [Features/Columns](#)
- [Community Hall](#)
- [Knowledge Bank](#)
- [About etown.com](#)
- [CE Links](#)

### NEW EPISODE!



**Interactive Knowledge Bank** 

### Ida: Your Interactive Decision Assistant



#### Choosing a digital camera

Picking out the right digital camera is a matter of finding out which features and benefits will work best for you. Answer a few questions for us, and we'll help you pick one that you'll like. There are 50 digital cameras to choose from so let's get started.

**How are you planning on using the images that you'll shoot with your new digital camera? (Check all that apply.)**

- Post them to a Web site.** I need enough picture quality for my shots to look good on screen, though I don't need to print them.
- Email them to friends and family.** I need a wide range of picture quality; some people may want to print the pictures I send.
- Make prints out of them.** I need an upper-level camera with the best possible picture resolution, because at some point, there'll be hard copies.

[Next question](#) 

### Picture quality

While no digital camera produces pictures with the same quality as a good film-based camera, the gap is closing, particularly with the advent of the new "megapixel" digital cameras, which feature more than a million pixels of image quality. However, for casual exchange of photos, or use on Web sites, much of this resolution is unnecessary -- screen resolution on a PC is only 75 pixels per inch.

Of course, the better your original image -- in other words, the higher the resolution it was shot with -- the better your final presentation is, whether it's on screen or in a hard-copy print. As you look through your choices, consider how much resolution you really need for your intended use. If you don't need megapixel performance, there's no reason to pay extra for it, particularly when there are excellent digital cameras around that will do fine for Web or email work.

### Search

  
 Products   
[Advanced Search](#)

QuickJump



Need expert advice?  
**Ask Ida!**

### LIVE HELP

 [CLICK HERE](#) to chat with our customer service



**Customer Service**  
24 hours a day  
7 days a week

**Toll Free**  
877-GO-ETOWN

### CHAT! with our experts

**Audio Talk**  
Played LOUD!  
with Michael Fremer  
8pm Tuesdays



Start

Quick

Book

Wine.c...

book

xterm

Shortcuts...

Cannot...

G:\pap...

emacs...

Calend...

etow...

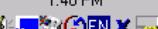
713tod...

Micros...

Micros...

Internet

1:40 PM





etown.com™

Customer Service | Your Account | Shopping Cart

July 17, 2000

etown.com consumer electronics  
**Summer Sale**Save Over 70%  
Click Here**Categories**

- [Home Theater](#)
- [Camcorders](#)
- [Home Audio](#)
- [Television](#)
- [Portable Tech](#)
- [Telecom](#)

**News / Views**

- [This Just In](#)
- [Reviews/Products](#)
- [Features/Columns](#)
- [Community Hall](#)
- [Knowledge Bank](#)
- [About etown.com](#)
- [CE Links](#)

**NEW EPISODE!**

Interactive Knowledge Bank

**Ida: Your Interactive Decision Assistant****Experience**

Your answers to the next 2 questions will help me determine which digital cameras I should recommend to you.

- I'm a casual user.** I like to shoot pictures, and I don't like to fuss with technology. The simpler my camera is, the better I like it.
- I know my way around a camera.** I'm interested in a digital camera with features that can make my work better and more enjoyable.
- I'm an avid photographer.** I want an advanced digital camera that can keep up with my ideas and provide me with the most creative options.

[Next question](#) **High-end features**

All digital cameras are capable of point-and-shoot simplicity, but some have more options and features than others. If you're a casual photographer, many of these features can appear intrusive, and in most shooting situations, you may not ever need them. However, if you're an experienced photographer, you'll want a model with an extensive feature set that includes manual overrides, so that you'll have control over exposure and focus for creative effects.

[Top of Page](#)**Search**
  
 Products 
[Advanced Search](#)

QuickJump

Need expert advice?  
**Ask Ida!****LIVE HELP**
[CLICK HERE](#) to chat with our customer service
 Customer Service  
24 hours a day  
7 days a weekToll Free  
877-GO-ETOWN**CHAT!** with our experts
**Audio Talk**  
 Played LOUD!  
 with Michael Fremer  
 8pm Tuesdays



etown.com™

Customer Service | Your Account | Shopping Cart

July 17, 2000

[The worlds largest Selection]

**Categories**

- [Home Theater](#)
- [Camcorders](#)
- [Home Audio](#)
- [Television](#)
- [Portable Tech](#)
- [Telecom](#)

**News / Views**

- [This Just In](#)
- [Reviews/Products](#)
- [Features/Columns](#)
- [Community Hall](#)
- [Knowledge Bank](#)
- [About etown.com](#)
- [CE Links](#)

**NEW EPISODE!**

Interactive Knowledge Bank [GET FLASH](#)

**Ida: Your Interactive Decision Assistant****Budget choice**

I have divided all the digital cameras into 4 budget categories. If you would like, I can exclude products that cost more than you are willing to spend.

**About how much would you consider spending on a new digital camera?**

- **Up to \$300.** In this price range you can expect to find a fairly basic digital camera with no zoom lens and standard picture resolution (under a megapixel). These cameras will produce images that are fine for posting to the Web or emailing, but do less well when they're printed on paper. *All of the 6 digital cameras in this range fit your expressed needs well.*
- **Up to \$600.** At the lower end of this price range you'll find some megapixel cameras that produce images that can be printed out at small sizes with acceptable results. Toward the high end, there are feature-rich cameras that produce fairly high-quality images that translate into sharp prints at sizes up to 4 x 6 inches. *All of the 24 digital cameras in this range fit your expressed needs well.*
- **Up to \$900.** If bigger-sized hard copies are what you're after, you'll probably have to step up to this price range. Cameras here generally have a zoom lens and a minimum resolution of 1.3 megapixels; some top the 2-megapixel mark. Most models in this category also have more flexibility in their operation, with more manual settings. *All of the 46 digital cameras in this range fit your expressed needs well.*
- **\$1000 or more.** This is "prosumer" territory -- the fine line between professional and consumer gear. Many models have 2-megapixel or better resolution, as well as plenty of manual options for more advanced users. Zoom is standard, and some models may have options for adding lenses (wide angle or telephoto). *All of the 50 digital cameras in this range fit your expressed needs well.*

[See initial recommendations](#) **Search**
  
 Products 
[Advanced Search](#)

QuickJump

**LIVE HELP**

[CLICK HERE](#)  
to chat with our customer service



**Customer Service**  
24 hours a day  
7 days a week

**Toll Free**  
877-GO-ETOWN

**CHAT!** with our experts

Audio Talk  
Played LOUD!  
with Michael Fremer  
8pm Tuesdays



Customer Service | Your Account | Shopping Cart

July 17, 2000

etown.com consumer electronics  
**Summer Sale**Save Over 70%  
Click Here**Categories**

- [Home Theater](#)
- [Camcorders](#)
- [Home Audio](#)
- [Television](#)
- [Portable Tech](#)
- [Telecom](#)

**News / Views**

- [This Just In](#)
- [Reviews/Products](#)
- [Features/Columns](#)
- [Community Hall](#)
- [Knowledge Bank](#)
- [About etown.com](#)
- [CE Links](#)

**NEW EPISODE!**

What is GPS?

**Interactive Knowledge Bank****Ida: Your Interactive Decision Assistant****Initial recommendations**

Here's my initial suggested shortlist of the products that best meet your needs, based on what you've told me so far. They all offer an LCD view screen, manual overrides, an optical viewfinder, a serial output connection, a video out connection, a built-in digital zoom, and are within your requested price range.

► **Olympus D450Z** [See etown.com Review](#) \$499

**Pros:** it can store 18 pictures at its highest resolution, it has 1280 x 960 pixels resolution, it uses SmartMedia to store pictures, and it has an optical zoom lens.

► **Olympus D460 Zoom** [Buy](#) \$499

**Pros:** it can store 18 pictures at its highest resolution, it has 1280 x 960 pixels resolution, it uses SmartMedia to store pictures, and it has an optical zoom lens.

► **Fuji MX1200** [See etown.com Review](#) [Buy](#) \$299

**Pros:** it can store 23 pictures at its highest resolution, it has 1280 x 960 pixels resolution, and it uses SmartMedia to store pictures. **Cons:** it doesn't have an optical zoom lens.

► **Nikon 800** [Buy](#) \$599

**Pros:** it has 1600 x 1200 pixels resolution, it has CompactFlash storage media, and it has an optical zoom lens. **Cons:** it can store only 8 pictures at its highest resolution.

**Top of Page**

I can refine these recommendations if you tell me more about your needs. I suggest **Optical zoom** as the next question to consider, or you can select the topic you wish:

[Next question](#)

Optical zoom



I can also show you a feature-by-feature comparison of any of the digital cameras listed on this page.

[Compare Products](#)**+ Other suitable products****Search**

Products

[Advanced Search](#)

QuickJump

Need expert advice?  
**Ask Ida!****LIVE HELP****CLICK HERE**

to chat with our customer service

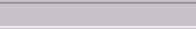


**Customer Service**  
**24 hours a day**  
**7 days a week**

**Toll Free**  
**877-GO-ETOWN**

**CHAT!** with our experts

Audio Talk  
Played LOUD!  
with Michael Fremer  
8pm Tuesdays



Quick Y:\book Wine.c... Y:\book xterm Shortc... Cannot... G:\pap... emacs... Calend... etown... 713tod... Micros... Micros...

1:45 PM





etown.com™

Customer Service | Your Account | Shopping Cart

July 17, 2000

# Change the way you think...

**Categories**

- [Home Theater](#)
- [Camcorders](#)
- [Home Audio](#)
- [Television](#)
- [Portable Tech](#)
- [Telecom](#)

**News / Views**

- [This Just In](#)
- [Reviews/Products](#)
- [Features/Columns](#)
- [Community Hall](#)
- [Knowledge Bank](#)
- [About etown.com](#)
- [CE Links](#)

**NEW EPISODE!**

What is GPS?  
[Interactive Knowledge Bank](#) [GET FLASH](#)

**Ida: Your Interactive Decision Assistant****Optical zoom**

Not every digital camera has the ability to zoom in for closeups. Some can, but only by digitally enlarging what the lens sees, a technique which is not as effective as an optical zoom. Some models have an actual optical zoom lens.

**How important is a zoom?**

- Not very.** I'm not shooting from any great distances, I don't need to spend extra on a zoom feature.
- A digital zoom is fine.** I don't need anything too extensive, but some kind of zoom feature would be useful to me.
- I need an optical zoom.** I need a the best possible zoom function on my digital camera.

[See Recommendations](#) **Optical and digital zoom**

A zoom lens brings the subject you're shooting closer -- a useful feature if you're at a distance from your subject. However, there's a big difference between a digital and optical zoom.

A digital zoom doesn't really zoom at all. What it does is electronically enlarge the information that the lens is seeing. While this does make the image larger (more close up), the image quality suffers from the digital manipulation.

An optical zoom is part of the lens itself -- there's no digital sleight-of-hand, and no degradation of the image, no matter how much you zoom. Because an optical zoom is hardware, as opposed to a digital zoom, which is done in software, a digital camera with an optical zoom is usually more costly.

[Top of Page](#)**Search**
  
 Products   
[Advanced Search](#)

QuickJump



Need expert advice?  
**Ask Ida!**

**LIVE HELP**

[CLICK HERE](#) to chat with our customer service



**Customer Service**  
24 hours a day  
7 days a week

**Toll Free**  
877-GO-ETOWN

**CHAT!** with our experts

Audio Talk  
Played LOUD!  
with Michael Fremer  
8pm Tuesdays





Customer Service | Your Account | Shopping Cart

July 17, 2000

Save Over 70%  
Click Here

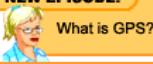
## Categories

- [Home Theater](#)
- [Camcorders](#)
- [Home Audio](#)
- [Television](#)
- [Portable Tech](#)
- [Telecom](#)

## News / Views

- [This Just In](#)
- [Reviews/Products](#)
- [Features/Columns](#)
- [Community Hall](#)
- [Knowledge Bank](#)
- [About etown.com](#)
- [CE Links](#)

## NEW EPISODE!



## Interactive Knowledge Bank



## Ida: Your Interactive Decision Assistant

## Recommendations



Here's my shortlist of the products that best meet your needs, based on the new information you gave me. They all offer an LCD view screen, manual overrides, an optical viewfinder, a built-in digital zoom, an optical zoom lens, and are within your requested price range.

► **Olympus D450Z** [See etown.com Review](#) \$499

**Pros:** it can store 18 pictures at its highest resolution, it has 1280 x 960 pixels resolution, it uses SmartMedia to store pictures, and it has a video out connection.

► **Olympus D460 Zoom** [Buy](#) \$499

**Pros:** it can store 18 pictures at its highest resolution, it has 1280 x 960 pixels resolution, it uses SmartMedia to store pictures, and it has a video out connection.

► **Fuji FinePix1400** [Buy](#) \$399

**Pros:** it has 1280 x 960 pixels resolution, it uses SmartMedia to store pictures, and it has a USB connection. **Cons:** it can store only 6 pictures at its highest resolution, it doesn't have a serial output connection, and it doesn't have a video out connection.

► **Nikon 800** [Buy](#) \$599

**Pros:** it has 1600 x 1200 pixels resolution, it has CompactFlash storage media, and it has a video out connection. **Cons:** it can store only 8 pictures at its highest resolution.

## Top of Page

I can refine these recommendations if you tell me more about your needs. I suggest **Shooting aids** as the next question to consider, or you can select the topic you wish:

[Next question](#) [Shooting aids](#)

I can also show you a feature-by-feature comparison of any of the digital cameras listed on this page.

[Compare Products](#)

## + Other suitable products

## Search

  
 Products   
 Search  
[Advanced Search](#)

QuickJump



Need expert advice?  
**Ask Ida!**

## LIVE HELP

[CLICK HERE](#)  
to chat with our customer service



Customer Service  
24 hours a day  
7 days a week

Toll Free  
877-GO-ETOWN

## CHAT!

with our experts  
Audio Talk  
Played LOUD!  
with Michael Fremer  
8pm Tuesdays



1:49 PM





etown.com™

Customer Service | Your Account | Shopping Cart

July 17, 2000



DVD movie rentals

**Categories**

- [Home Theater](#)
- [Camcorders](#)
- [Home Audio](#)
- [Television](#)
- [Portable Tech](#)
- [Telecom](#)

**News / Views**

- [This Just In](#)  
[Reviews/Products](#)  
[Features/Columns](#)  
[Community Hall](#)  
[Knowledge Bank](#)  
[About etown.com](#)  
[CE Links](#)

**NEW EPISODE!**

What is GPS?

**Interactive Knowledge Bank** [GET FLASH](#)

**Ida: Your Interactive Decision Assistant****✓ Nikon 800 - \$599.00 (msrp)**

Given the information you have provided, the Nikon 800 is one of my top recommendations. Click the link to go directly to a question that will explain the feature and help you decide if that feature makes sense for you!

**Pros: Its advantages include:**

- it has an [LCD view screen](#).
- it has manual overrides.
- it has an [optical viewfinder](#).
- it has 1600 x 1200 pixels [resolution](#).
- it has a [serial output](#) connection.
- it has [CompactFlash](#) storage media.
- it has a [video out connection](#).
- it has a built-in digital zoom.
- it has an [optical zoom](#) lens.

**Cons: Possible disadvantages include:**

- it can [store](#) only 8 pictures at its highest resolution.
- it doesn't have a [USB connection](#).

\*The Nikon 800 is available for [online purchase](#).

[Return to Recommendations](#) [Show full specification](#) [Next Question](#) **Search**
  
 Products 

 Advanced Search  
  
 QuickJump 

 Need expert advice?  
**Ask Ida!**
**LIVE HELP**
 CLICK HERE  
 to chat with our customer service

 Customer Service  
 24 hours a day  
 7 days a week

 Toll Free  
 877-GO-ETOWN
**CHAT! with our experts**
 Audio Talk  
 Played LOUD!  
 with Michael Fremer  
 8pm Tuesdays

[Modify Comparison List](#) | [Return to Product List](#)

## Your Product Comparisons

Products are compared by key features. When a product lacks a feature, the feature appears in pale grey. To view a full Product Profile, click on the corresponding model number. To compare different products, click on Modify Comparison List (above).

Digital Cameras:

Digital Cameras:

Digital Cameras:



|                                       |                                      |                                 |
|---------------------------------------|--------------------------------------|---------------------------------|
| Fuji                                  | Nikon                                | Olympus                         |
| <a href="#">Model No: FinePix1400</a> | <a href="#">Model No: 800</a>        | <a href="#">Model No: D450Z</a> |
| 1.2-megapixel digital camera          | CoolPix 2.1-megapixel digital camera | 1.2-megapixel digital camera    |
| 1 year parts & labor                  | 1 year parts & labor                 | 1 year parts & labor            |
| List Price: \$399.00                  | List Price: \$599.00                 | List Price: \$499.00            |
| Low Price: \$329.00                   | Low Price: \$489.99                  | ---                             |
| High Price: \$329.00                  | High Price: \$498.00                 | ---                             |

**\$ Buy Now!**[Find Local Retailer](#)**\$ Buy Now!**[Find Local Retailer](#)**Find Local Retailer**

320 x 240

320 x 240

320 x 240

640 x 480

640 x 480

640 x 480

800 x 600

800 x 600

800 x 600

1024 x 760

1024 x 760

1024 x 760

1200 x 1024

1200 x 1024

1200 x 1024

1536 x 1024

1536 x 1024

1536 x 1024

1944 x 1000

1944 x 1000

1944 x 1000



|                                  |                                  |                                   |
|----------------------------------|----------------------------------|-----------------------------------|
| 1760 x 1160                      | 1760 x 1160                      | 1760 x 1160                       |
| 1152 x 864                       | 1152 x 864                       | 1152 x 864                        |
| 1280 x 960                       | 1200 x 960                       | 1280 x 960                        |
| 1600 x 1200                      | 1600 x 1200                      | 1600 x 1200                       |
| 1000 x 1200                      | 1000 x 1200                      | 1000 x 1200                       |
| 6 pictures at highest resolution | 8 pictures at highest resolution | 18 pictures at highest resolution |
| 40 pictures at lowest resolution | 50 pictures at lowest resolution | 122 pictures at lowest resolution |
| Built-in flash                   | Built-in flash                   | Built-in flash                    |
| Compact body                     | Compact body                     | Compact body                      |
| Digital zoom                     | Digital zoom                     | Digital zoom                      |
| Floppy disk storage              | Floppy disk storage              | Floppy disk storage               |
| IEEE 1394 output                 | IEEE 1394 output                 | IEEE 1394 output                  |
| LCD view screen                  | LCD view screen                  | LCD view screen                   |
| Megapixel resolution             | Megapixel resolution             | Megapixel resolution              |
| SLR-type body                    | SLR-type body                    | SLR-type body                     |
| Optical zoom lens                | Optical zoom lens                | Optical zoom lens                 |
| Optical viewfinder               | Optical viewfinder               | Optical viewfinder                |
| Manual overrides                 | Manual overrides                 | Manual overrides                  |
| Serial output                    | Serial output                    | Serial output                     |
| Storage media: CompactFlash      | Storage media: CompactFlash      | Storage media: CompactFlash       |
| Storage media: SmartMedia        | Storage media: SmartMedia        | Storage media: SmartMedia         |
| Storage media: Memory Stick      | Storage media: Memory Stick      | Storage media: Memory Stick       |
| USB output                       | USB output                       | USB output                        |
| Video output                     | Video output                     | Video output                      |
| Windows software included        | Windows software included        | Windows software included         |
| Macintosh software included      | Macintosh software included      | Macintosh software included       |

# Knowledge-Based Recommender

- Case-Based Example with Navigation Interface
- FindMe Systems (e.g., Entrée)



# Entree Results

The Los Angeles restaurant you chose is:

## Chinois On Main

2709 Main St (bet Rose Ave. & Ocean Park Blvd.), Santa Monica, 310-392-9025

Pacific New Wave

\$30-\$50

Extraordinary Decor, Extraordinary Service, Near-perfect Food, Hip Place To Be, On the Beach, Great for People Watching, Parties and Occasions, Weekend Brunch, Weekend Lunch, Fabulous Wine Lists

We recommend:

## Yoshi's Cafe

3257 N. Halsted St (Belmont Ave.), Chicago, 312-248-6160

Asian, Japanese, French (New)

\$30-\$50

Extraordinary Decor, Extraordinary Service, Near-perfect Food, Need To Dress, Prix Fixe Menus, Quiet for Conversation, Very Busy - Reservations a Must, Romantic, Good Out of Town Business, Fabulous Wine Lists, Game, Parking/Valet

less \$\$

Less \$\$

nicer

cuisine

traditional

creative

livelier

quieter

 *Entree Results*

**For a cheaper restaurant than:**

|   |           |
|---|-----------|
| <b>Yoshi's Cafe</b>                                       |           |
| 3257 N. Halsted St. (Belmont Ave.), Chicago, 312-248-6160 |           |
| Asian, Japanese, French (New)                             | \$30-\$50 |

**We recommend:**

|   |            |
|---|------------|
| <b>Lulu's</b> <small>(map)</small>  |            |
| 626 Davis St. (bet. Chicago & Orrington Aves.), Evanston, 708-869-4343  |            |
| Japanese, Asian   | below \$15 |
| Good Decor, Excellent Service, Excellent Food, Creative, No Reservations, Weekend Brunch, Wheelchair Access, Long Drive |            |

*less \$\$      nicer      cuisine*  
*traditional      creative      livelier      quieter*

Figure 3: Navigation using the "Less \$\$" tweak

# More Generally

- Case-Based Approaches (Knowledge, Database, etc.) are often most helpful for ephemerally-personalized experiences
  - Shopping – suggest similar relevant items
    - Compare with collaborative – suggest items that are co-purchased or co-browsed
  - Content – suggest similar stories
- Case-Based recommendations are often easier to explain to the user

# Challenges and Drawbacks

- Content-Based Techniques in general ...
  - Depend on well-structured attributes that align with preferences (consider paintings)
  - Depend on having a reasonable distribution of attributes across items (and vice versa)
  - Unlikely to find surprising connections (e.g., chili peppers or lemon with chocolate)
  - Harder to find complements than substitutes

# Some take-away lessons

- Many ways to recommend based on content (product attributes)
  - Long-term: build profile of content preferences
  - Shorter-term: build database of cases; navigate
- Content-based techniques work without a large set of users (but need item data)
- Good at finding substitutes; good at helping navigate for a purchase; good explainability

# Moving Forward

- Next Lectures
  - For programmers: Introduction to LensKit
  - For everyone: deeper dive into content profiles, content retrieval and filtering
- Later this Module
  - Programming deep dive; guest lectures on case-based and knowledge-based; survey of tools for content recommending

# 3-1: Introduction to Content-Based Recommenders

# 3-2: Introduction to LensKit

# Introduction

- The remaining programming assignments will be using LensKit
- This lecture will provide an overview of LensKit
  - Project goals
  - Core APIs
  - Layout of LensKit
  - Architecture of a Recommender

# LensKit Project Goals

An open-source toolkit for *building, researching,*  
*and studying* recommender systems.

# Goals in Detail

- *Building* — usable to build working, real-world recommender systems
- *Researching* — platform for recommender research in algorithms, evaluation, and user experience
- *Studying* — tool for education about recommender system use/implementation

# Core APIs

- Recommender – provides access to the rest of the APIs
- RatingPredictor – predicts user ratings for items
- ItemScorer – generates user-personalized scores for items
- ItemRecommender – recommends items for users

# LensKit Modules

- lenskit-api – core APIs (previous slide)
- lenskit-data-structures – data structures used by LensKit and its APIs
- lenskit-core – foundational utilities and infrastructure for LensKit
- lenskit-eval – evaluation toolkit
- lenskit-{knn,svd,slopeone} – algorithm implementations

# Fastutil

- Java generics interact badly with primitives
- Fastutil provides primitive versions of Java collections
- e.g. LongList is List<Long> without boxing
- LensKit uses these extensively for performance

# LensKit Data Structures

- SparseVector – map long ids (e.g. users or items) to values.
- Cursor – iterator with a `close()` method
- ScoredId – an id (user or item) with a score
- Collection utilities
- Symbols

# SparseVector

- Sparse, ordered map from long keys to double values
- Valid keys fixed when vector is created
- SparseVector is read-only abstract class
- Mutable and immutable implementations
- Supports linear algebra operations
  - Add, subtract, multiply, etc. only on mutable vectors
  - Modify the vector in-place

# API in depth: ItemRecommender

```
public interface ItemRecommender {  
    List<ScoredId> recommend(long user);  
    List<ScoredId> recommend(long user, int n);  
    List<ScoredId> recommend(long user, int n,  
        Set<Long> candidates,  
        Set<Long> exclude);  
}
```

# API in depth: ItemScorer

```
public interface ItemScorer {  
    double score(long user, long item);  
    SparseVector score(long user,  
                      Collection<Long> items);  
    void score(long user,  
              MutableSparseVector scores);  
}
```

# API in depth: ItemScorer

```
public interface ItemScorer {  
    double score(long user, long item);  
    SparseVector score(long user,  
                      Collection<Long> items);  
void score(long user,  
          MutableSparseVector scores);  
}
```

- Abstract base class leaves MSV implementation abstract, implements others using it.

# Data Access in LensKit

- Primary data type is *Event* – an interaction of a user with an item
  - Rating is a subclass of Event
- Event data comes from data access objects (DAOs)
- Master dao is *EventDAO*
- Other interfaces provide different access
  - UserEventDAO – get events for a user
  - ItemEventDAO – get events for an item

# Using LensKit

- Create a LensKit configuration
- Create or configure a data access object
- Configure algorithm and parameters
- Build recommender from configuration
- Get individual recommender components from recommender

# Structure of a LensKit algorithm

- LensKit algorithm implementation usually implements ItemScorer
- Generic RatingPredictor, ItemRecommender implementations work with many scorers
- Algorithm consists of many components
- Goal: reconfigurable algorithms with many reusable parts
- Dependency injection organizes everything

# Dependency Injection

- Dependency Injection is a design pattern for organizing classes that use each other
- A class depending on another class has an instance of its dependency *injected* (often via constructor parameter)
- Dependency *injector* automates the process of instantiating many classes built around DI.
- LensKit uses Grapht, which is much like Guice.

# Example: TopNRecommender

# Common Algorithm Components

- Item scorer
- Rating predictor
- Item recommender
- Baseline scorer
- Normalizers

# Example: User-item mean scorer

```
EventDAO myDao = new SimpleFileRatingDAO(file);
LenskitConfiguration config =
    new LenskitConfiguration();

config.bind(EventDAO.class)
    .to(myDao);
config.bind(ItemScorer.class)
    .to(UserMeanItemScorer.class);
config.bind(UserMeanBaseline.class, ItemScorer.class)
    .to(ItemMeanRatingItemScorer.class);

Recommender rec = LenskitRecommender.build(config);
```

# Putting it together: lenskit-hello

# More Information

- APIs and components documented in JavaDoc
- BitBucket wiki has more long-form documentation and introductions
- Coursera forums

# 3-2: Introduction to LensKit

# 3-3: TFIDF and More!

# Learning Objectives

- To understand the problem that requires a weighting for search or filtering
- To understand TFIDF weighting in detail, and how it is used in both search and filtering
- To understand the range of variants and alternatives to TFIDF
- To appreciate the similarities and differences between content filtering and search

# The search problem ...

- Why do primitive search engines fail?
- What would a primitive search engine do?
  - Return all documents that contain search terms?
  - More frequent occurrence ranked higher?
- At a minimum, need to consider two factors
  - Term frequency may be significant
  - Not all terms equally relevant
- Actually, much harder than this, more later ...

# TFIDF weighting

- Term Frequency \* Inverse Document Frequency
- Term Frequency =
  - Number of occurrences of a term in the document (can be a simple count)
- Inverse Document Frequency =
  - How few documents contain this term
  - Typically  
 $\log (\# \text{documents} / \# \text{documents with term})$

# What does TFIDF do?

- Automatic demotion of stopwords, common terms
- Promotes core terms over incidental ones

But where does it fail?

- If core term/concept isn't actually used (much) in document (e.g., legal contracts)
- Poor searches (other techniques for that)

# How does TFIDF apply to CBF?

- TFIDF concept can be used to create a profile of a document/object
  - A movie could be described as a weighted vector of its tags (details next lecture)
- These TFIDF profiles can be combined with ratings to create user profiles, and then matched against future documents

# Variants and Alternatives

- Some applications use variants on TF
  - 0/1 boolean frequencies (occurs above threshold)
  - Logarithmic frequencies ( $\log (tf+1)$ )
  - Normalized frequency (divide by document length)
- BM25 (aka Okapi BM25) is a ranking function used by search engines:
  - Includes frequency in query, in document, number of documents, length
  - Variants with different weights: BM11, BM15, ...

# Actually much harder, as we said

- Phrases and n-grams
  - “computer science” != “computer” and “science”
  - Adjacency
- Significance in Documents
  - Titles, headings, ...
- General Document Authority
  - Pagerank and similar approaches
- Implied Content
  - Links, usage ...

# Take-Away and Moving Forward

- You should
  - Understand TFIDF and why it is needed
  - Also understand its limitations
- Next
  - Building and applying content profiles

# 3-3: TFIDF and More!

# 3-4: Content-Based Recommenders in Detail

# Learning Objectives

- To how to build content-based recommenders based on TFIDF concepts, including:
  - Computing vectors to describe items
  - Building profiles of user preference
  - Predicting user interest in items
- To understand key variants in implementing CBF recommenders, and their strengths and weaknesses

# Key concept: Keyword Vector

- The universe of possible keywords defines a content space
  - Each keyword is a dimension
  - Each item has a position in that space; that position defines a vector
  - Each user has a taste profile (or more than one) that is also a vector in that space
  - The match between user preference and items is measured by how closely the two vectors align
  - May want to limit/collapse keyword space (e.g., stem and stop)

# Useful reference

- Vector Space Model (originally conceived for queries, indexing)
  - [http://en.wikipedia.org/wiki/Vector\\_space\\_model](http://en.wikipedia.org/wiki/Vector_space_model)
  - Salton, Wong, and Yang (1995) “A Vector Space Model for Automatic Indexing,” CACM 18:11.

# Where choices come into play ...

- Representing an item through a keyword vector:
  - Simple 0/1 (keyword applies or doesn't)
  - Simple occurrence count
  - TFIDF, most commonly:
    - # occurrences in doc \* log (# docs / #docs with term)
  - Other variants that include factors such as document length
  - Eventually, this vector is often normalized

# Consider the movie/tag case ...

- Do we consider tags to be yes-or-no?
  - Actor (we don't really get a measure for how much "Tom Hanks" a movie has)
  - Descriptive (is how often a tag is applied a proxy for how relevant/significant the feature is?)
- Do we care about IDF?
  - Actor (are infrequent actors more significant than stars?)
  - Descriptive (is "prison scene" more significant than "car chase" or "romance"?)

# From Items to User Profiles (1)

- Vector Space model conflates liking with importance
  - Works well in some applications (query terms), not as much in others (I like Hollandaise sauce a lot, but don't actually care much if it is in a dish I'm ordering)
  - Consider how this may play out with movie keywords ...

# ...to User Profiles (2)

- How do we accumulate profiles?
  - Add together the item vectors?
    - Do we normalize first?
      - Do we believe an item with a longer vector is more descriptive of preferences? Then maybe no.
      - Do we think all items should be the same weight? Then maybe yes.
    - Do we weigh the vectors somehow?
      - Could use ratings for weight ...
      - Could use currency, confidence ...

# ...to User Profiles (3)

- How do we factor in ratings?
  - Simply unary – aggregate profiles of items we rated without weights
  - Unary with threshold – only put items above a certain rating into our profile (but all likes are equal) – we often think 3.5 in MovieLens
  - Weight, but positive only – higher weight for things with higher scores
  - Weight, and include negative also – negative weight for low ratings (normalize rating scale)

# ...to User Profiles (3)

- How do we update profiles (new ratings)?
  - Don't – just recompute each time (wasteful)
  - Weight new/old similarly – keep track of total weight in profile and mix in new rating (linear combination)
    - Special case for changed rating; subtract old
  - Decay old profile and mix in new (e.g., may decide that profile should be dominated by most recent 50 movies – formula might be  $0.95 * \text{old} + 0.5 * \text{new}$ , in normalized form)

# Computing Predictions ...

- Prediction is the cosine of the angle between the two vectors (profile, item)
- This is the dot-product of normalized vectors, or if you prefer, the dot product divided by the product of the two lengths
- Cosine ranges between -1 and 1 (0 and 1 if all positive values in vectors) – closer to 1 is better.
- Top-n, or scale for rating-scale predictions.

# Strengths of this Approach

- Entirely content-based
- Understandable profile
- Easy computation
- Flexibility – can integrate with query-based systems, case-based approaches

# Challenges

- Figuring out the right weights and factors
  - Is more more, or just reiteration of the same
  - How to deal with ratings

# Limitations

- This is a highly simplified model, cannot handle interdependencies
  - I like Sandra Bullock in Action movies, but Meg Ryan in Romantic Comedy movies
  - I like comedies with violence, and historical documentaries, but not historical comedies or violent documentaries

# Take-aways

- Content-based filtering based on assessing the content profile of each item; can be done from metadata or user tagging
- User profiles built by aggregating profiles of items rated/consumed, possibly with a weighting scheme
- Evaluate unrated items by matching item profile against user profile: vector cosine

# Moving Forward

- Next Lecture
  - Survey of content filtering tools
- Assignments
  - Written: Using content filtering; design exercise
  - Programming: Adding a basic content filtering module to LensKit
- Looking further forward
  - After collaborative filtering, some of these concepts return when we look at SVD/LSI

# 3-4: Content-Based Recommenders in Detail

# 3-5 Tools for Content-Based Filtering

# Introduction

- You probably don't want to build from scratch
  - Especially for prototyping
- Many tools are available for building recommenders
- This lecture: a brief survey of a few of them (particularly open-source ones)

# LensKit

- Provides high-level recommendation framework
- Does not currently provide content-based recommender implementations
- Helps with the plumbing for small- or medium-scale applications

# Search Software

- Many content-based filtering algorithms are search algorithms (incl. TF-IDF)
- Existing search software useful to build recommenders

# Apache Lucene

- Open-source Java search package
- Very widely used
- Provides document indexing w/ arbitrary fields and fast search
- Several relevance and ranking algorithms
- Good, out-of-the-box performance

# Using Lucene

1. Create an index
2. Add ‘document’ representations of items
3. Construct queries
4. Ask for results (will be scored)

# Building the Index

```
IndexWriterConfig config = /* configure */ ;  
Directory dir = FSDirectory.open(indexFile);  
IndexWriter w = new IndexWriter(dir, config);  
for (ItemInfo item: getItems()) {  
    Document doc = new Document();  
    doc.add(new Field("title", item.title));  
    doc.add(new Field("tags", item.tags));  
    w.add(doc);  
}  
  
w.close();
```

# Finding Similar Items

```
IndexSearcher idx = getIndexSearcher();
IndexReader reader = idx.getIndexReader();
Document itemDoc = findItemDoc(idx, item);
MoreLikeThis mlt = new MoreLikeThis(reader);
String[] fields = {"title", "tags"};
mlt.setFieldNames(fields);
Query q = mlt.like(docid);
TopDocs results = idx.search(q, n + 1);
```

# Related Projects

- SOLR provides search server (with REST API) on top of Lucene
- PyLucene is Python implementation
- Lucy is in C w/ bindings for other langs
- Lucene.NET
- Semantic Vectors provides LSI for Lucene

# Other Search Software

- Lemere (C++, Univ. of Maryland)
- Xapian (C++)
- Any search package is a valuable tool

# Case-based Reasoning

- jCOLIBRI is a Java-based CBR framework

# Machine Learning Toolkits

- Recommendation algorithms can be seen as a special-case of machine learning
- Machine learning libraries can be useful to implement recommenders
  - Weka
  - Apache Mahout
  - Milk, SciPy/NumPy (Python)
  - R, Matlab

# Just Scratch the Surface

- There are many toolkits
  - Some general-purpose
  - Some special-purpose
- These are just a few, that may be helpful particularly for CBF recommenders
- Discuss others in the forums

# 3-5 Tools for Content-Based Filtering

```
IndexWriterConfig config = /* configure */ ;
Directory dir = FSDirectory.open(indexFile);
IndexWriter w = new IndexWriter(dir, config);
for (ItemInfo item: getItems()) {
    Document doc = new Document();
    doc.add(new Field("title", item.title));
    doc.add(new Field("tags", item.tags));
    w.add(doc);
}
w.close();

IndexSearcher idx = getIndexSearcher();
IndexReader reader = idx.getIndexReader();
Document itemDoc = findItemDoc(idx, item);
MoreLikeThis mlt = new MoreLikeThis(reader);
String[] fields = {"title", "tags"};
mlt.setFieldNames(fields);
Query q = mlt.like(docid);
TopDocs results = idx.search(q, n + 1);
```

# 4-1: Introduction to User-User Collaborative Filtering

# Learning Objectives

- To understand the intuition and history of the user-user collaborative filtering algorithm
- To review the basic ideas and assumptions (and therefore limitations) behind the algorithm.

# Historical Reflection ...

- 1992: Tapestry and seeds of ACF
- 1994-1995 Early Automated CF Systems
  - GroupLens
  - Ringo/HOMR
  - Video Recommender

# Common Characteristics

- Collection of Ratings
- Measure of Inter-User Agreement
  - Correlation, Vector Cosine
- Personalized Recommendations/Predictions
  - Weighted Combinations of Others' Ratings
- Tweaks to make things work right ...
  - Neighborhood limitations
  - Normalization
  - Dealing with limited co-ratings

# Implementation Issues

- Given  $m$  users and  $n$  items:
  - Computation can be a Bottleneck
    - Correlation between two users is  $O(n)$
    - All correlations for a user is  $O(mn)$
    - All pairwise correlations is  $O(m^2n)$
    - Recommendations at least  $O(mn)$
  - Lots of ways to make more practical
    - More persistent neighborhoods ( $m \rightarrow k$ )
    - Cached or incremental correlations

# Core Assumptions/Limitations

- Why does this work?
  - Let's break it down ...
- Assumption: Our past agreement predicts our future agreement
  - Base Assumption #1: Our tastes are either individually stable or move in sync with each other
  - Base Assumption #2: Our system is scoped within a domain of agreement

# So What Happened?

- GroupLens -> Net Perceptions -> GroupLens
- RINGO -> Agents Inc. -> Firefly Networks
- Industry Acceptance of ACF
  - Pressure to innovate more efficient algorithms

# Moving Forward

- Next Lectures
  - Breaking down the core algorithm
  - Tuning and tweaks: normalization, neighborhood size, and more
- Later this Module
  - Explanations, trust, reputation
  - Programming user-user CF

# 4-1: Introduction to User-User Collaborative Filtering

# 4-2: Breaking Down User-User Collaborative Filtering

# Learning Objectives

- To understand the details of the user-user collaborative filtering algorithm

# Key Reference

- An Algorithmic Framework for Collaborative Filtering by Herlocker, Konstan, Borchers, and Riedl (Proc. SIGIR 1999)

# Moving Forward

- Next
  - Normalization, neighborhood size, and more ...

# 4-2: Breaking Down User-User Collaborative Filtering

# 4-3: User-User Variations and Tuning

# Introduction

- Previous lectures
  - User-user collaborative filtering
  - How user-user CF works
- This lecture
  - Customizations and design decisions

# Learning Objectives

- Know the main implementation decisions to make in user-user CF
- Understand different options and why they might be selected
- Have a best-practice starting point for configuring a user-user CF recommender

# Overview

- Selecting Neighborhoods
- Scoring Items from Neighborhoods
- Normalizing Data
- Computing Similarities
  - Algorithms
  - Tweaks
- Additional Options

# Overview

- **Selecting Neighborhoods**
- Scoring Items from Neighborhoods
- Normalizing Data
- Computing Similarities
  - Algorithms
  - Tweaks
- Additional Options

# Selecting Neighborhoods

- All the neighbors
- Threshold similarity or distance
- Random neighbors
- Top- $N$  neighbors by similarity or distance

# How Many Neighbors?

- In theory, the more the better
  - If you have a good similarity metric
- In practice, noise from dissimilar neighbors decreases usefulness
- Between 25 and 100 is often used
  - 30–50 often good for movies
- Fewer neighbors → lower coverage

# Overview

- Selecting Neighborhoods
- **Scoring Items from Neighborhoods**
- Normalizing Data
- Computing Similarities
  - Algorithms
  - Tweaks
- Additional Options

# Scoring Items from Neighborhoods

- Average
- Weighted average
- Multiple linear regression

Weighted average is common, simple, and works well

# Overview

- Selecting Neighborhoods
- Scoring Items from Neighborhoods
- **Normalizing Data**
- Computing Similarities
  - Algorithms
  - Tweaks
- Additional Options

# What's wrong with data?

- Users rate differently
- Some rate high, others low
- Some use more of the scale than others
- Averaging ignores these differences
- Normalization compensates for them

# Mean-centering

- Subtract user mean prior to computing
- Re-add when needed

# z-score normalization

- Mean-center, and divide by standard deviation
- Normalizes for the spread across the scale
- Small additional gain in prediction accuracy over mean-centering

# Other normalizations

- Subtract item mean
- Subtract item-user mean

# Overview

- Selecting Neighborhoods
- Scoring Items from Neighborhoods
- Normalizing Data
- **Computing Similarities**
  - Algorithms
  - Tweaks
- Additional Options

# Computing Similarities

- Last time: Pearson correlation

$$s(a, u) = \frac{\sum (r_{ai} - \mu_a)(r_{ui} - \mu_u)}{\sqrt{\sum (r_{ai} - \mu_a)^2} \sqrt{\sum (r_{ui} - \mu_u)^2}}$$

- Usually only over ratings in common
- User normalization not needed
- Spearman rank correlation is Pearson applied to *ranks*
  - Hasn't been found to work as well

# Problem: what about little data?

- Suppose users have 1 rating in common
- Pearson correlation is 1
- Are the users really similar?

# Solution: significance weighting

- Weight similarity by confidence
- Simple approach: multiply by  $1/\min(n, 50)$ 
  - < 50 common ratings: scaled down by # of common ratings
  - $\geq 50$  common ratings: unscaled
- Can also do Bayesian damping

# Vector Similarity

- Compute cosine of user vectors in rating space

$$sim(a, u) = \frac{\mathbf{a} \cdot \mathbf{u}}{\|\mathbf{a}\| \|\mathbf{u}\|}$$
$$= \frac{\sum r_{ai} r_{ui}}{\sqrt{\sum r_{ai}^2} \sqrt{\sum r_{ui}^2}}$$

- With user-mean norm: Pearson correlation!

# Self-weighting Similarity

- Cosine has built-in significance weighting
- Weights proportionally to ratio of common ratings & total ratings (roughly)
- Similar effect as using overall  $\sigma_u$  instead of just over common ratings in Pearson

# Overview

- Selecting Neighborhoods
- Scoring Items from Neighborhoods
- Normalizing Data
- Computing Similarities
  - Algorithms
  - Tweaks
- **Additional Options**

# Clustering

- Cluster users
- Pick user's cluster to generate predictions
- Doesn't work particularly well

# Pre-computation

- Expensive
- Users move as their ratings change

# Baseline Configuration

- Top  $N$  neighbors ( $\sim 30$ )
- Weighted averaging
- User-mean or z-score normalization
- Vector similarity over normalized ratings

# Conclusion

- There are a variety of configuration points
- Current research has suggested some that work well
- Next module will discuss evaluation methods you can use to find good options for your application

# 4-3: User-User Variations and Tuning

$$\begin{aligned}
 \text{sim}(a, u) &= \frac{\vec{a} \cdot \vec{u}}{\|\vec{a}\| \|\vec{u}\|} \\
 &= \frac{\sum_i \hat{r}_{ai} \cdot \hat{r}_{ui}}{\sqrt{\sum_i \hat{r}_{ai}^2} \sqrt{\sum_i \hat{r}_{ui}^2}} \quad \hat{r}_{ai} = r_{ai} - M_a \\
 &= \frac{\sum (r_{ai} - M_a)(r_{ui} - M_u)}{\sqrt{\sum (r_{ai} - M_a)^2} \sqrt{\sum (r_{ui} - M_u)^2}}
 \end{aligned}$$

Scale by  $N$

$$\frac{|R_a \cap R_u|}{|R_a| |R_u|}$$

$$\text{sim}(a, u) = \frac{\sum_{i=1}^n (r_{ai} - \mu_a)(r_{ui} - \mu_u)}{\sqrt{\sum_{i=1}^n (r_{ai} - \mu_a)^2} \sqrt{\sum_{i=1}^n (r_{ui} - \mu_u)^2} + K}$$

$$\frac{x \cdot y}{\sqrt{x^2} \sqrt{y^2}} = 1$$

$$\text{sim}(a, u) \cdot \frac{1}{\min(50, |R_a \cap R_u|)}$$

$$\mu + \hat{\mu}_i + \hat{\mu}_u$$

$$\hat{\mu}_i = \frac{\sum_u (r_{ui} - \mu)}{\#\text{of ratings}(i)}$$

$$\hat{\mu}_u = \frac{\sum_i (r_{ui} - \mu - \hat{\mu}_i)}{\#(u)}$$

$$s(a, i) = \frac{\sum_{u \in \mathcal{U}} r_{ui} \cdot \text{sim}(a, u)}{\sum_u \text{sim}(a, u)}$$

$$\frac{\sum_u (r_{ui} - \mu_u) \cdot \text{sim}(a, u)}{\sum_u \text{sim}(a, u)} + \mu_a$$

$$z_{ui} = \frac{r_{ui} - \mu_u}{\sigma_u}$$

$$s(a, i) = \frac{\sum_u z_{ui} \cdot \text{sim}(a, u)}{\sum_u |\text{sim}(a, u)|} \cdot \sigma_a + \mu_a$$

$$P = \frac{\sum (r_{ai} - M_a)(r_{ui} - M_u)}{\sqrt{\sum (r_{ai} - M_a)^2} \sqrt{\sum (r_{ui} - M_u)^2}}$$

$$\frac{\sum (r_{ai} - M_a)(r_{ui} - M_u)}{\sqrt{\sum (r_{ai} - M_a)^2} \sqrt{\sum (r_{ui} - M_u)^2}}$$

# 4-4: Explaining Recommendations

# Learning Objectives

- To understand the variety of ways that explanations are made in recommender systems, and how users react to them.
- To understand how explanations can fit into a recommender system
- To recognize pitfalls in explanation – and particularly the difference between explanation and persuasion

# Key Reference

- Explaining Collaborative Filtering Recommendations by Herlocker, Konstan, and Riedl (Proc. CSCW 2000)
  - Several dozen more recent papers look at specific aspects – transparency, trust, computing efficient explanations, and more ...

# What is an Explanation?

- Additional data to help users understand a specific recommendation
  - Separate from an explanation of how the system works as a whole
  - Some explanations are confidence scores
- Often a glimpse inside the computation
  - More detail than a simple prediction score ...
    - Predictions don't distinguish normal from weird distributions
- Sometimes tied to ability to edit profile to improve recommendations ...

# Let's take a look at Herlocker ...

- Key lessons
  - Simplicity is key – users didn't like “correlation,” statistical terms, or being overwhelmed with data
  - Simple visualizations work well – histogram, table
  - Supporting information valued – historical success, attribute-linked data, associations
- One key mistake
  - We didn't actually measure explanation effectiveness, but rather persuasiveness

# Explanations in Practice

- We saw Amazon.com earlier
- “Why?” is a compelling anchor
- Tell me more provides user control
- Be careful, though – overwhelming the user is rarely a good strategy
- Explanations not only for user-user CF
- Mostly open area: explaining “lists” vs. items

# Take-Away

- You've now seen several types of explanation:
  - More data on the prediction itself
  - Other data about the item
  - Simplified presentations of relevant data
  - Past performance data
  - Supporting statistics
- You now have some experience with which explanations work with ordinary users ...

# Moving Forward

- Next
  - Interviews related to trust, reputation, influence of bad ratings, alternative ways to focus on trusted sources or limit unknown ones
  - Assignments related to user-user recommendation
  - Next module focuses on evaluation – will also provide new metrics that could help with explanation!

# 4-4: Explaining Recommendations

# 5-1: Introduction to Evaluation of Recommender Systems

# Goals for Today

- To understand ways of evaluating the “goodness” of a recommendation, and of a recommender algorithm or system
  - Accuracy metrics
  - Error metrics
  - Decision-support metrics
  - User and Usage-centered metrics
- To understand how predictions and recommendations (including top-n) are evaluated
- To understand retrospective and live approaches to evaluation

# Why a whole module on evaluation?

- Zillions of algorithms, but which to pick?
- Lessons from commercial experience
- Lessons from the Netflix Challenge
- Lessons from (and for) the research community

# A Historical Look

- The early days
  - Accuracy and error measures:
    - MAE, RMSE, MSE
  - Decision-support metrics:
    - ROC AUC, Breese score, later precision/recall
  - Error meets decision-support/user experience:
    - “Reversals”
  - User-centered metrics:
    - Coverage, user retention, recommendation uptake, satisfaction

# A Commercial Look

- Nobody cared about accuracy ...
  - The supermarket recommender
- Lift, cross-sales, up-sales, conversions
- Led to thinking about different measures anchored not only to user experience, but recommender goals

# Moving Forward ...

- Lots of new metrics developed as researchers looked at tuning for specific purposes:
  - More sophisticated top-n / rank metrics
  - Serendipity
  - Diversity
- More systematic evaluation of the recommender as a whole (not just the recommendations)

# Theme 1: Prediction vs. Top-N

- Key distinction:
  - Prediction is mostly about accuracy, possibly decision support; focused locally
  - Top-N is mostly about ranking, decision support; focused comparatively

# Theme 2: More than Just Metrics

- Even simple evaluations are hard ...
  - How to calculate Mean Absolute Error
    - Easy to compute error of a single prediction
    - Average across predictions or across users?
    - How to handle lack of coverage?
- Comparative evaluation is even harder ...
  - Proper baseline
  - Different coverage, etc.

# Theme 3: Unary Data

- Many of the metrics we present are designed specifically for evaluating data with a multi-point rating scale (e.g., 1-5).
- Some measures don't work well at all for unary data (e.g., purchase data)
- Special coverage of unary evaluation ...

# Theme 4: Dead vs. Live Recs?

- Retrospective (dead data) evaluation looks at how recommender would have predicted or recommended for items already consumed/rated.
- Prospective (live experiment) evaluation looks at how recommendations are actually received.
- Fundamental differences ...

# Looking forward ...

- This module includes:
  - Lectures on the major types of evaluation, and on how to conduct a rigorous evaluation
  - A “rant” on when evaluations may be meaningless
  - Assignments focused on conducting evaluation, both by hand and on a large scale with LensKit
- Going forward, evaluation should be part of your toolkit ...

# 5-1: Introduction to Evaluation of Recommender Systems

# 5-2: Basic Accuracy Metrics

# Goals for Today

- To understand how to compute
  - MAE -- Mean absolute error
  - MSE -- Mean squared error
  - RMSE -- Root mean squared error
- To understand variations on how these may be computed
- To understand where accuracy metrics are useful in general, and the relative merits of each of these three

# A little intuition

- Error metrics are usually computed using a “leave one out” methodology
  - Cover up a rating, and try to predict it
- Warning: sometimes this is hard, and evaluators take short cuts (e.g., leave 10% out).

# Mean Absolute Error (MAE)

- What is error?
  - Divergence of prediction from actual opinion (rating)
  - P-R
- Absolute error removes direction
  - $|P - R|$
  - Why? Because two wrongs don't make a right!
- MAE = Average ( $|P - R|$ )
  - $$\frac{\sum_{\text{ratings}} |P - R|}{\# \text{ ratings}}$$

# Mean Squared Error (MSE)

- Why Squared Error?
  - Removes sign – avoids need for absolute value
  - Penalizes large errors more than small
- $$\frac{\sum_{\text{ratings}} (P - R)^2}{\# \text{ ratings}}$$
- One disadvantage – squared error is not on an intuitive scale ...

# Root Mean Squared Error (RMSE)

- $\sqrt{\frac{\sum_{ratings} (P - R)^2}{\# ratings}}$

# Hold on a moment ...

- We glossed over the summation
  - Usual model – average over all ratings
  - Alternative model – average over user averages
- What's the difference
  - What if one user has 3000 ratings and another 10?
- Advice – consider looking at both – understand what you're comparing to

# Comparing Different Algorithms

- What to do when computing MAE in different cases:
  - Remember, must be same data set/scale
  - If coverage is different (different set of user/item pairs for which predictions are available, two choices):
    - Check against common subset
    - Supplement algorithm with default for full coverage

# Reflections ...

- In general, all the error metrics move together (good replacements for each other)
- Squared may matter for large scales with some algorithms that have occasional huge errors, but other measures may catch that better
- Benefit – lots of published MAE data for public datasets
- Drawback – error can be dominated by irrelevant parts of the item space

# Looking forward ...

- Next, we look at decision-support metrics

# 5-2: Basic Accuracy Metrics

# 5-3: Basic Decision Support Metrics

# Goals for Today

- To understand the concept of “decision support” metrics
- To learn a set of decision-support metrics, including:
  - Error rate and Reversals
  - Precision, Recall, MAP
  - Receiver operating characteristic
- To understand the usefulness and limitations of these metrics

# What is “Decision Support”

- Measure how well a recommender helps users *make good decisions*
  - Good decisions are about choosing “good” items and avoiding “bad” ones
- For predictions: 4\* vs. 2.5\* worse than 2.5\* vs. 1\*
- For recommendations, top of list is what matters most.

# Errors and Reversals

- What is an “error?”
  - Ad hoc measure of wrong predictions
  - E.g., determine that  $3.5-5^*$  = good,  $1-2.5^*$  = bad
    - Error is when a good movie (for a user) gets a bad prediction (or vice versa)
  - Can also be used for top-n – every time a bad movie appears in the top-n, it is an error.
  - Usually reported as total number (compared between algorithms), average error rate per user, etc.
  - Not widely used in research
- Reversals are large mistakes – e.g., off by 3 points on a 5-point scale
  - Intuition is that these are likely really bad – lead to loss of confidence
  - Again, reported as total or average rate

# Precision and Recall

- Information Retrieval Metrics
  - Precision is the percentage of selected items that are “relevant”
    - $P = \frac{N_{rs}}{N_s}$
  - Recall is the percentage of relevant items that are selected
    - $R = \frac{N_{rs}}{N_r}$

# Precision and Recall (2)

- Different Goals
  - Precision is about returning mostly useful stuff
    - Not wasting user time
    - Assumption is that there is more useful stuff than you want
  - Recall is about not missing useful stuff
    - Not making a bad oversight
    - Assumption is that you have time to filter through results to find the key result you need
  - When these two goals are in balance, F-metrics

$$\bullet \quad F_1 = \frac{2PR}{P+R}$$

# Precision and Recall (3)

- Problem #1 with precision/recall
  - Need ground truth for all items
    - But if we had ground truth, why bother with a recommender
  - Ways this is addressed
    - Fake precision/recall by limiting to rated items
      - Common – results in interesting biases
    - Human-rating experiments that compute precision/recall over some random subset

# Precision and Recall (4)

- Problem #2 with precision/recall
  - Covers entire data set – not targeted on top-recommended items
    - precision/recall inherently about “full query”
  - Addressed through P@n, R@n
    - Precision@n is the percentage of the top-n items that are “good”:  $P@n = \frac{N_{r@n}}{n}$
    - Some have proposed computing this as an average over a set of experiments with 1 “hit” and a large number of presumed misses
    - Recall@n is effectively the same

# Mean Average Precision (MAP)

- In IR, MAP averages over both multiple queries and over position in top-n retrieval
  - $MAP = \frac{\sum_{q=1}^Q AveP(q)}{Q}$  where  $AveP = \frac{\sum_{k=1}^n P(k)*rel(k)}{\# \text{ relevant docs}}$
  - More intuitively, this is computing an estimate of the area under the precision-recall curve, and then averaging across queries
- In recommender systems, this has been adapted in several ways, most commonly across users. Unfortunately, not standarized.
  - Useful comparing algorithm variants; for other comparisons, need to be really careful ...

# Receiver Operating Characteristic

- The ROC curve is a plot of the performance of a classifier or filter at different thresholds. It plots true-positives against false positives:
  - [http://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic](http://en.wikipedia.org/wiki/Receiver_operating_characteristic)
- In recommender systems, the curve reflects trade-offs as you vary the prediction cut-off for recommending (vs. not).
- Area under the curve is often used as a measure of recommender effectiveness

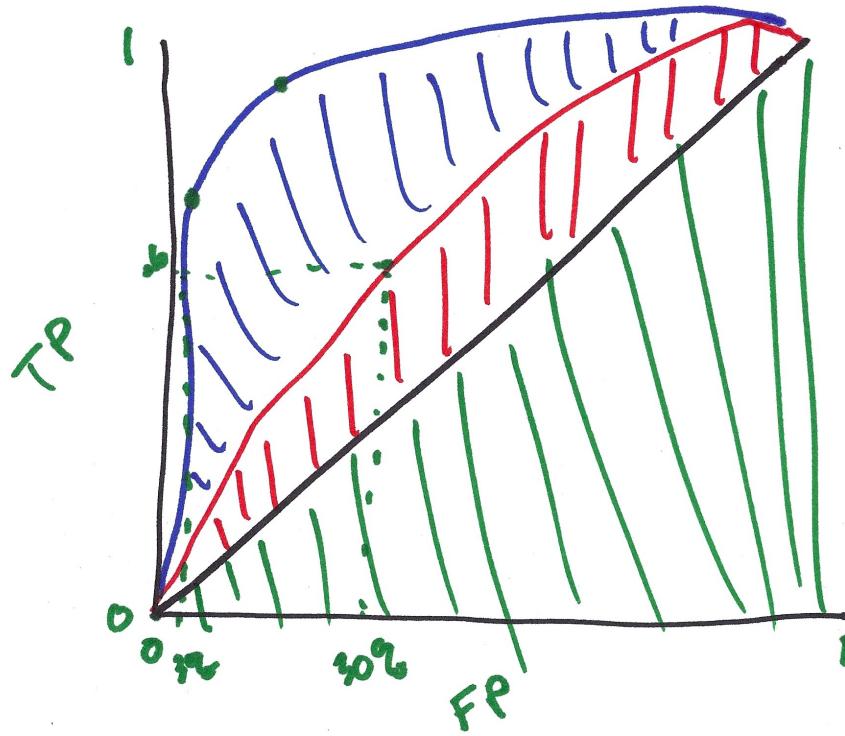
# Reflections ...

- Once again, all of these metrics tend to correlate highly with each other (good replacements for each other)
- Precision@n and overall precision are perhaps the most widely used (and easily understood)
- ROC provides insight if the goal is to tune the recommender's use as a filter, or identify “sweet spots” in its performance
- None of these metrics overcome the problem of being based on rated items only (and the inherent variation that comes from this limitation)

# Looking forward ...

- Next, we look at rank metrics, then a bit of a rant on hidden-data evaluation, and then a broader set of metrics to look at business relevance ...

# 5-3: Basic Decision Support Metrics



# 5-4: Rank Metrics

# Introduction

- Previous lectures
  - Evaluation methodology
  - Prediction accuracy metrics
  - Decision support metrics
- This lecture
  - Measuring how good a recommender is at ranking

# Learning Objectives

- Understand the basic idea of measuring rank accuracy vs. decision support or prediction accuracy
- Understand how to apply and interpret common rank accuracy metrics

# Metric Families

- Prediction accuracy: how well does the recommender estimate preference?
- Decision support: how well does the recommender do at finding good things?
- Rank accuracy: how well does the recommender estimate *relative* preference?
  - Putting items in order by preference

# Overview

- Mean Reciprocal Rank
- Spearman Rank Correlation
- Discounted Cumulative Gain
- Fraction of Concordant Pairs

# Mean Reciprocal Rank

- Reciprocal rank:  $1/i$ , where  $i$  is the rank of the first ‘good’ item
- Similar to precision/recall
  - P/R measures how good recommender is at only being relevant (precision) and finding things (recall)
  - RR measures how far you have to go to find something good
- MRR is just average over all test queries

# Spearman Rank Correlation

- Pearson correlation over ranks
- Punishes misplacement

$$\frac{\sum_i (r_1(i) - \mu_1)(r_2(i) - \mu_2)}{\sqrt{\sum_i (r_1(i) - \mu_1)^2} \sqrt{\sum_i (r_2(i) - \mu_2)^2}}$$

# What's wrong with Spearman?

- Punishes all misplacement equally
- However: we don't care as much low-down
  - swapping 1 and 3: bad
  - swapping 11 and 13: not nearly so bad
- Goal: weight things at the top of the list more heavily

# Discounted Cumulative Gain

- Measure *utility* of item at each position in the list
- Discount by position, so things at front are more important
- Normalize by total achievable utility
- Result is Normalized Discounted Cumulative Gain (nDCG)

# nDCG: The Formula

$$\text{DCG}(r) = \sum \text{disc}(r(i))u(i)$$

$$\text{nDCG}(r) = \frac{\text{DCG}(r)}{\text{DCG}(r_{\text{perfect}})}$$

- nDCG is in range [0,1]; 1 is perfect ranking
- Measures fraction of potential utility achieved

# Utility

- Rating
- 1/0 (for purchases/clicks/views)
  - Be careful – next lecture

# Discount

- $1/(\log_2 r(i))$  is common
  - no discount for 2<sup>nd</sup> item
- Half-life utility (Breese, 1996) has good theoretical basis  $2^{\frac{r(i)-1}{\alpha-1}}$ 
  - Exponential decay
  - Idea: users are exponentially less likely to click each successive item in list
  - Measurement: expected utility

# Fraction of Concordant Pairs

- What fraction of pairs are in the correct relative order?
- Tests pairwise accuracy

# Conclusion

- Several metrics to measure recommender's ability to order items
- nDCG increasingly common; MRR also used
- Next lecture: problems with missing-data evaluations

# 5-4: Rank Metrics

# 5-5: The Fallacy of Hidden Data Evaluation: A Rant!

# In Summary

- Recommenders are about helping people find new stuff they'll like
- Hidden-data evaluations penalize algorithms that *actually find new stuff!*
  - They're convenient, and can rule out many bad algorithms ...
- In the end, for real relevance, we need to look at user experience (see 5-10)

# 5-5: The Fallacy of Hidden Data Evaluation: A Rant!

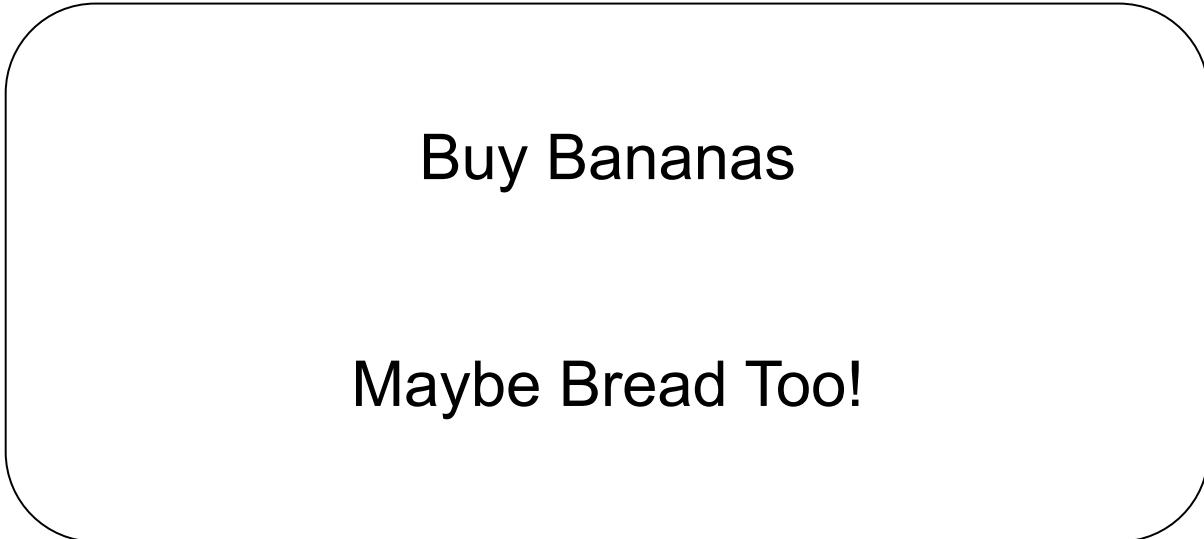
# 5-6: More Metrics

# Goals for Today

- To understand metrics that relate more closely to business goals and/or user experience, specifically:
  - Coverage
  - Diversity
  - Serendipity
- To understand why it may be worth trading off accuracy for other purposes ...

# A couple of stories ...

- The perfect supermarket recommender ...



Buy Bananas

Maybe Bread Too!

# Ziegler's Inspiration ...

- Let's see what books Amazon.com recommends that I buy ...

# Coverage

- Coverage is the measure of the percentage of products for which a recommender can make a prediction
  - Or a prediction that's personalized
  - Or a prediction above a confidence threshold
- Computed as a simple percentage
  - Inconsistent averaging over user/unrated item
  - Easiest is to “hide” every item and compute for entire data set

# Use of Coverage

- Directly relevant in cases where predictions are displayed
  - What percentage of movies will I get a star-score for?
- Often used as “background” metric when comparing top-n recommenders
  - Some have extended to ask what percentage of items will appear in *someone’s* top-n
- Business interest: reach entire catalog ...

# Diversity

- Measure of how different the items recommended are
  - Applied to a top-n list
- Start with a pairwise similarity metric
  - E.g., Ziegler used book categories, others have used tags, keyword vectors
- Intra-list similarity is the average pairwise similarity, lower score is higher diversity

# Diversification ...

- Common approach is to penalize/remove the items from the top-n list that are too similar to prior items already recommended (never touch #1)
  - Replace with  $n+1^{\text{st}}$  or later items – first ones that don't exhibit too high a similarity
  - Diversification factor limits how many substitutions will be made

# Alternatives to Diversification

- Clustering approaches allow “diversification through bundling”
- Scatter-gather interfaces allow user-controlled diversification
- Business goal: don’t turn away customers who are not currently interested in a narrow portion of your catalog

# Serendipity

- Definition: “the occurrence and development of events by chance in a happy or beneficial way”
- In recommender systems: surprise, delight, not the expected results
- Several ways to operationalize, such as:
  - $serend = \frac{1}{N} \sum_{i=1}^N \max(\Pr(s_i) - Prim(s_i), 0) * isrel(s_i)$
  - Key concept – need prior “primitive” estimate of obviousness, one such metric is overall popularity.

# Serendipity in Practice

- Don't need an overall metric to increase serendipity
  - Simply downgrade items that are highly popular (or otherwise obvious)
  - This tends to require experimentation and tuning
  - May use diversification factor approach, or a constant down-weighting by popularity
- Business goal – get people to consume less popular items

# Business Objectives and Metrics

- Users of recommenders have a much broader set of objectives than the metrics we've discussed:
  - Immediate lift
  - Net lift (subtract out cost of returns)
  - Time to next transaction
  - Long-term customer value (lifetime value)
  - Referrals
  - And much more ....

# Wrap up ...

- Metrics can address “fuzzier” goals such as producing a diverse, delightfully surprising set of recommendations; or assessing whether recommendations go deep into the “long tail” of the catalog and not just a few oft-recommended items.
- Experimental data shows that these objectives may even be worth sacrificing some accuracy.

# Looking forward ...

- Now that we have an extensive set of metrics, a couple of sessions on how to use them rigorously, and how to apply them to special cases ...

# 5-6: More Metrics

# 5-7: Experimental Protocols for Rating Data

# Introduction

- We've discussed several evaluation metrics
- We now turn to experimental protocol design
  - How do we structure an evaluation using these metrics?

# Learning Objectives

- Understand basic structure of a crossfold recommender evaluation
- Be able to design a plausible, repeatable evaluation using best practices
  - For rating or yes/no data

# Goal of Offline Evaluation

- To *estimate* the recommender's quality
  - High-throughput evaluation
  - Answer important research questions
- Often cannot answer if recommender really works
  - User-based evaluation needed
  - Link to business metrics is weak

# Background

- Offline protocols inspired by related research areas
  - Machine learning
  - Information retrieval

# Machine Learning

- Hidden data
  - Hold out some data, try to predict/classify it
- Cross-validation
  - Split data into partitions, hold out each in turn
  - Average results
  - Mitigates effects of split in results
- Measure score or classification accuracy

# Information Retrieval

- Measure accuracy in providing results for queries with known results
- Uses known preference judgements

# Adapting to Recommenders

- Use ratings/purchases/clicks as relevance judgements or ground truth
- Measure recommendations or predictions

# Basic Structure

- Partition data set into  $k$  partitions
- For  $i = 1$  to  $k$ 
  - train on all sets other than  $i$
  - test on set  $i$
- What  $k$  to use?
  - Large values → more training data
  - Small values → more efficient
  - 5 and 10 are common

# Splitting data

- Split ratings
- Split users
  - Allows more control for measuring expected user experience
- Split items
  - Rarely, if ever, done

# Splitting users

- Split user ratings randomly
  - Very common
  - Use to compare with existing results
- Split user ratings by time
  - More accurate simulation of user experience
  - Results often worse
- Best, but expensive: only train on ratings before time of test rating

# Using log data

- Log data often unary (clicked, purchased), nothing known about absent items
- Basic structure is the same
- More discussion in next lecture

# Good Practice

- Split users into  $k$  partitions (5 is common)
- Split user ratings by time
  - Use random to compare with previous results
- Include user query ratings in train data
- Document your protocol carefully
  - So you can run it again
  - So others can compare

# 5-7: Experimental Protocols for Rating Data

# 5-8: Unary Data Evaluation

# Introduction

- We've talked a lot about ratings data
- Some metrics are applicable for unary
  - P/R and friends
- This time: unary data
  - Often studied under ‘implicit feedback’
  - Unary data is positive-only (purchase, like)

# Implicit Feedback Data

- Many recommender contexts have no ratings or other explicit data
- Often data owners have non-unary data
  - Like vs. saw but didn't like
  - Clicked vs. saw but skipped
- W/ negative examples, can just do standard eval
- Use more data if you have it

# Unary Data

- We often don't have negatives
- Anyone using a data dump
  - song plays (don't know didn't play)
  - click logs
- Intrinsic to certain domains/tasks
  - research papers
  - physical store purchases

# Problems

- No negative examples
- How do we know if the recommender is wrong?
  - Or if the user just didn't know about the item?
- Put differently: how do we avoid punishing the recommender for doing its job?

# Metrics

- Precision/Recall/MAP
  - but is ‘bad’ really bad?
- MRR (Mean Reciprocal Rank)
  - still gets pushed down
- % At or Before Rank
  - histogram of raw data for MRR
- nDCG
  - first item may still be misjudged

# Mitigation strategies

- Synthesize unary data to get negatives
  - e.g. ratings,  $\geq 3.5$  stars is ‘like’
  - only recommend from rated data
- Limit domain of recommendation
  - recommend from good + N unknown
  - limits likelihood of good-but-unknown
    - these items are probably excluded

# Best we can do

- These evaluations are the best we have
- So use them
  - But be aware of limitations when reporting
- Look for alternatives
  - User testing
  - Try to get negative data
- Corroborate with additional evidence

# Promising Directions

- One-sided classification
- New metrics and protocols (e.g. clarity)

# Conclusion

- Evaluating recommenders is hard
- Offline evaluation doubly so
- We don't have great methods right now
- Be aware of problems when making claims
  - Both in research and industry

# 5-8: Unary Data Evaluation

# 5-10: User-Centered Evaluation

# Learning Objectives

- To understand the goals behind user-centered evaluation, and where it complements or replaces offline evaluations
- To become familiar with a variety of mechanisms for user-centered evaluation of recommender systems

# Why Evaluate with Users?

- Metrics such as MAE, top-N Precision, diversity, and the like only tell part of the story
  - The real question involves user preference and behavior
  - Don't know how users balance different attributes, different objectives, contexts
  - Need to understand difference between retrieval and recommendation
  - Most of all – user behavior is complex

# A Spectrum of Techniques

- Usage Logs
- Polls, Surveys, Focus Groups
- Lab (and online Lab) Experiments
- Field Experiments and Trials
- Other techniques as well ... often an area to consult an expert in user studies, HCI, etc.

# Usage Logs

- Means for evaluating use of features
  - Success of particular recommenders
    - Very useful with mixed hybrids – track separate success rates
  - Interface issues
- Potential for measuring retrospective accuracy, etc.

# Polls, Surveys, Focus Groups

- Can be useful for assessing overall desires, context, usage patterns
  - Be careful: hard to create good surveys
  - Techniques for combining responses, identifying underlying factors
  - Surveys often useful in conjunction with other techniques
- Don't confuse information gathering with selling!

# Lab and Online Lab Experiments

- Careful controls, but decontextualized
- Which recommendation list do you prefer?
- Rate these recommendations/lists on the following attributes (familiarity, accuracy, believability, interest, ...)
- Likert-scale questions common
- Be careful about question ordering (e.g., general to specific, if goal is to get both)

# Field Experiments and A/B Tests

- Beauty of this domain is ability to try variants and measure results ...
  - What do you want to measure:
    - Immediate Behavior (recommendations taken, options changed, etc.)
    - Longer-term Behavior (return rate, change in purchases, referrals, ...)
    - Subjective Responses (prompts/surveys at strategic times)
- A bit about the culture of massive A/B testing

# Take-Away

- No substitute for real user-centered testing
- Need to design tests around goals
  - Different methods can achieve different results
- Need an implementation and users for field experiments and usage logs, but all others can be done with simulated systems

# Moving Forward

- Next Modules
  - This is the last lecture on Evaluation
  - Returning to algorithms – first item-item collaborative filtering, then dimensionality-reduction algorithms

# 5-10: User-Centered Evaluation

# 6-1: Introduction to Item-Item Collaborative Filtering

# Learning Objectives

- To understand the motivation, history, and intuition behind item-item CF algorithms
- To gain a basic understanding of the algorithm idea, preparing you to master the details later this module
- To understand some of the practical strengths and weaknesses of the algorithm

# Motivation

- User-User CF was great, except ...
- Issues of Sparsity
  - With large item sets, small numbers of ratings, too often there are points where no recommendation can be made (for a user, for an item to a set of users, etc.)
  - Many solutions proposed here, including “filterbots”, item-item, and dimensionality reduction

# Motivation (2)

- Computational performance
  - With millions of users (or more), computing all-pairs correlations is expensive
  - Even incremental approaches were expensive
  - And user profiles could change quickly – needed to compute in real time to keep users happy

# The Item-Item Insight

- Item-Item similarity is fairly stable ...
  - This is dependent on having many more users than items
    - Average item has many more ratings than an average user
    - Intuitively, items don't generally change rapidly – at least not in ratings space (special case for time-bound items)
- Item similarity is a route to computing a prediction of a user's item preference

# A little more detail ...

- Two step process:
  - Compute similarity between pairs of items
    - Correlation between rating vectors
      - co-rated cases only (only useful for multi-level ratings)
    - Cosine of item rating vectors
      - can be used with multi-level or unary ratings
    - Adjusted cosine (normalize each user's ratings)
      - to adjust for differences in rating scales
    - Some use conditional probability (unary)
  - Predict user-item rating
    - Weighted sum of rated “item-neighbors”
    - Linear regression to estimate rating

# Item-Item Top-N

- Item-Item similarity model can be used to compute top-N directly:
  - Simplify model by limiting items to small “neighborhoods” of  $k$  most-similar items (e.g., 20)
  - For a profile set of items, compute/merge/sort the  $k$ -most similar items for each profile item
    - Straightforward matrix operation from Deshpande and Karypis

# Benefits of Item-Item

- It actually works quite well
  - Good MAE performance on prediction; good rank performance on top-N
- Efficient implementation
  - At least in cases where  $|U| \gg |I|$
  - Benefits of precomputability
- Broad applicability and flexibility
  - As easy to apply to a shopping cart as to a user profile

# Core Assumptions/Limitations

- Item-item relationships need to be stable ...
  - Mostly just a corollary of stable user preferences
  - Could have special cases that are difficult (e.g., calendars, short-lived books, etc.)
  - Many of these issues are general temporal issues
- Main limitation/complaint: lower serendipity
  - This is a user/researcher complaint, not fully studied; intuition is clear

# Moving Forward

- Next Lectures
  - Breaking down the core item-item algorithm
  - Looking at the special cases of unary/binary ratings
  - Programming item-item (for programmers)

# 6-1: Introduction to Item-Item Collaborative Filtering



# 6-2: Item-Item Algorithm

# Introduction

- We're now into the 2<sup>nd</sup> major personalized algorithm: item-item CF
- This lecture will discuss the algorithm in more detail
- Also: design space and performance implications

# Structure of Item-Item CF

- Pre-compute item similarities over all pairs of items
- Look for items similar to those the user likes
  - Or has purchased
  - Or has in their basket

# Components

- Item similarity function
- Model builder
- Neighborhood selection strategy
- Item score aggregation function

# Item Similarities

- Usually use cosine similarity between item rating vectors
- Often normalize user ratings first
  - Subtract user mean
  - Subtract item mean

$$sim(i, j) = \frac{\sum_{u \in U(i) \cap U(j)} \hat{r}_{ui} \hat{r}_{uj}}{\sqrt{\sum_u \hat{r}_{ui}^2} \sqrt{\sum_u \hat{r}_{uj}^2}}$$

# Scoring Items

- Score is driven by item
- For each item to score:
  - Find similar items the user has rated
  - Compute weighted average of user's ratings

$$p_{ui} = \frac{\sum_{j \in N} sim(i, j) r_{uj}}{\sum_{j \in N} |sim(i, j)|}$$

# Picking Neighbors

- Score formula had a neighborhood  $N$
- Neighbors are usually  $k$  most similar items
  - That the user has rated
- Good value of  $k$  important
  - $k$  too small  $\rightarrow$  inaccurate scores
  - $k$  too large  $\rightarrow$  too much noise (low-similarity items)
  - $k=20$  often works well

# Building the Model

- Pre-compute similarities for all pairs of items
  - Item stability makes similarity pre-computation feasible
- Naïvely:  $O(|I|^2)$ 
  - If symmetric: only need to compute one direction
  - Sometimes can skip pairs

# Truncating the Model

- Don't need to keep the whole  $I^2$  model
- But need enough neighbors to find neighbors at score time
  - Since user hasn't rated everything, need  $M \gg k$  neighbors per item in model
- Balance memory use with accuracy and coverage
  - Mild runtime impact, if neighbors are sorted

# Tuning the model

- Tune using cross-validation
- Need to find good values for
  - baseline and normalization
  - similarity function (or just use cosine)
  - neighborhood size  $k$
  - model size  $M$
  - sometimes similarity is damped, but often doesn't help much

# Conclusion

- Item-item is efficient and straightforward
- A few parameters need tuning for specific data, domain
- Next: more tweaks
  - applying to unary data (implicit feedback)
  - repurposing and hybridization

# 6-2: Item-Item Algorithm

# 6-3: Item-Item on Unary Data

# Introduction

- We've talked about item-item over rating data
- Also works well on unary data (implicit feedback)
  - clicks
  - plays
  - purchases
- But some tweaks are needed

# Data Representation

- Rating values: user-item rating matrix
- Need some matrix to represent data
  - Logical (1/0) user-item ‘purchase’ matrix
  - Purchase count matrix
- Problem: what is a 0?
  - We just ignore that for item-item

# Data Normalization

- Standard mean-centering not meaningful
- But we can normalize user vectors to unit vectors
  - Intuition: users who like many items provide less information about any particular pair
- Could also consider: logging counts

# Computing Similarities

- Cosine similarity still works
- Can also use conditional probability
  - see Deshpande and Karypis paper

# Aggregating Scores

- Weighted average works for non-binary
  - counts
- For binary (0/1), just sum neighbor similarities
  - fixed neighborhood size means this isn't unbounded

$$\text{score}(u, i) = \sum_{j \in N} \text{sim}(i, j)$$

- Neighborhood selection unchanged (most similar)

# Conclusion

- Item-item basically works for unary data
- A few tweaks to algorithm components needed to make it well-behaved
- Test variants with your data/context
  - Evaluation tools we talked about last module help with this

# 6-3: Item-Item on Unary Data

Cosine

$$\text{sim}(i, j) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\| \|\vec{j}\|}$$

$U(i)$  - users who bought  $i$

Cond Prob

$$\text{sim}(i, j) = P(j|i) = \frac{P(i, j)}{P(i)} = \frac{|U(i) \cap U(j)|}{\frac{n}{|U(i)|}}$$
$$= \frac{|U(i) \cap U(j)|}{|U(i)|}$$

$$-1 \leq \text{sim} \leq 1$$

$$\frac{P(i, j)}{P(i) \cdot P(j)^\alpha} \quad \alpha$$
$$0 < \alpha < 1$$

# 6-4: Item-Item Hybrids and Extensions

# Introduction

- We've now introduced item-item CF
- Item-item CF is very flexible
- This time: hybrids and extensions

# Hybrid Recommenders

- Combine 2 or more algorithms
- General technique, not specific to any one algorithm
- Similar to stacking or boosting in machine learning

# Hybrid Techniques

- Weighting – combine algorithm scores
  - Can be extended with feature-weighted coefficients
- Switching – switch algorithms
- Mixed – mix output from diff. algorithms
- Use one algorithm as input to another
- See Burke reading for more discussion

# Feature-Weighted Linear Stacking

- Winning NetFlix Prize algorithm
- Linear combination of 100+ algorithms
- Coefficients depend on user/item features
  - Relative weight of algorithms shifts for different recommendation contexts

# Extending Item-Item

- Item-item is good for extending directly
- Simple parts with well-defined interfaces provide a lot of flexibility
- It's easy to understand what extensions do

# Example: User Trust

- Goal: incorporate user trustworthiness into item relatedness computation
  - User's global reputation, not per-user trust
- Solution: weight users by trust before computing item similarities
- High-trust users have more impact
- Massa and Avesani. 2004. ‘Trust-Aware Collaborative Filtering for Recommender Systems’

# Extension: Papers and PageRank

- Recommending research papers: useful to consider items as users who purchase the paper's citations
  - Same idea can apply to web pages
- Goal: incorporate paper 'importance' into recommender
- Solution: weight paper user vectors by the paper's PageRank (or HITS hub score)
- Ekstrand et al., 2010. Automatically Building Research Reading Lists.

# Restructuring: Item-Item CBF

- Basic item-item algorithm structure doesn't care how similarity is computed
- So why not use content-based similarity?
- Resulting algorithm really isn't a collaborative filter
- But it can work pretty well!
- Example: using Lucene to compare documents as neighborhood & similarity function

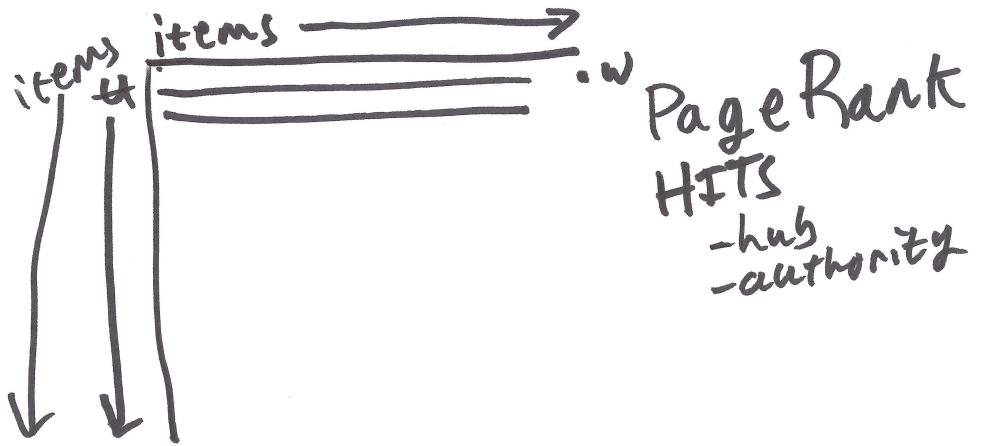
# Restructuring: Deriving Weights

- Item-item compares individual item pairs
- Alternative approach: infer coefficients from data
  - Find coefficients  $w_{i,j}$  that minimize RMSE
  - Learn coefficients with standard machine learning / optimization algorithm (gradient descent)

# Conclusion

- Single algorithms aren't only recommender solution
  - Many deployed algorithms are hybrids
- Item-item CF is flexible and versatile
- Many interesting recommenders can be built by reconfiguring it

# 6-4: Item-Item Hybrids and Extensions



$$p(u,i) = a_1 p_1(u,i) + a_2 p_2(u,i) \dots + b$$

$$p(u,i) = f_1(u,i) p_1(u,i) + f_2^{(u,i)} p_2(u,i) \dots + b$$

# Readings

## Module 1 (week 1)

- [Deconstructing Recommender Systems](#) (Lecture 1-4)
- [Setting Up LensKit](#)

## Module 2 (weeks 2–3)

- Lecture 2-4
  - [How Not to Sort by Average Rating](#)
  - [Hacker News ranking algorithm](#) (see also the [discussion with the creator](#))
  - [Reddit's news ranking algorithm](#) (as of 2010)
  - [Reddit's comment ranking algorithm](#) (as of 2011)

## Module 3 (weeks 4–5)

- [Anatomy of a LensKit Recommender](#)

## Module 4 (weeks 6–7)

### Required Readings

-  [Explaining collaborative filtering recommendations](#)  
**Jonathan L. Herlocker, Joseph A. Konstan, John Riedl**  
 CSCW '00 Proceedings of the 2000 ACM conference on Computer supported cooperative work, 2000
- [Collaborative Filtering Recommender Systems](#)

### Suggested Readings

- [The Influence Limiter](#)

## Module 5 (weeks 8–9)

### Required Readings

- [A Survey of Accuracy Evaluation Metrics of Recommendation Tasks](#), by Gunawardana and Shani
-  [Being accurate is not enough: how accuracy metrics have hurt recommender systems](#)  
**Sean M. McNee, John Riedl, Joseph A. Konstan**  
 CHI EA '06 CHI '06 Extended Abstracts on Human Factors in Computing Systems, 2006

### Suggested Readings

- [Netflix Recommendations: Beyond the 5 stars](#) (referenced in lecture 5-5)

-  [Evaluating collaborative filtering recommender systems](#)  
**Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, John T. Riedl**  
 ACM Transactions on Information Systems (TOIS), 2004
- [Explaining the user experience of recommender systems](#), by Knijnenburg et al.
-  [Improving recommendation lists through topic diversification](#)  
**Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, Georg Lausen**  
 WWW '05 Proceedings of the 14th international conference on World Wide Web, 2005

## Additional Resources

- [Performance prediction and evaluation in Recommender Systems: an Information Retrieval perspective](#) — Alejandro Bellogín's Ph.D thesis, analyzing some of the biases affecting offline recommender evaluation and how they might be mitigated.

## Module 6 (weeks 10–11)

### Required Readings

-  [Item-based collaborative filtering recommendation algorithms](#)  
**Badrul Sarwar, George Karypis, Joseph Konstan, John Riedl**  
 WWW '01 Proceedings of the 10th international conference on World Wide Web, 2001
- [Item-Based Top-N Recommendation Algorithms](#)
- [Hybrid Recommender Systems: Survey and Experiments](#), by Burke. (PDF available from the [author's publications pages](#))

### Supplemental Readings

- Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights ([available from the authors](#)).
- Ekstrand et al., 2010. [Automatically Building Research Reading Lists](#).
- Ekstrand et al., 2014. [User Perceptions of Differences in Recommender Algorithms](#) — the paper containing the results of the study on the ways in which recommenders differ in their output that Dr. Konstan mentioned in the supplementary video.

## Module 7 (weeks 12–13)

- [Incremental SVD-Based Algorithms for Highly Scalable Recommender Systems](#)
- [Netflix Update: Try This at Home](#) — Simon Funk's description of FunkSVD.

### Supplemental Readings

- [Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model](#)
- [Latent semantic models for collaborative filtering](#) (PLSI paper; unfortunately, we don't have an free version to point you to)
- Probabilistic Matrix Factorization (available from the [author's web site](#))

## Module 8 (week 14)

- [Recommendations as Personalized Learning to Rank](#)

# Setting Up LensKit

Most of the programming assignments for the Introduction to Recommender Systems course will be done in LensKit, an open-source toolkit for building and studying recommender systems. This document describes what you need to download and configure in order to use LensKit.

## Requirements

You will need the following to complete the LensKit programming assignments: Java Development Kit (we recommend the latest release of Java 7)

- Apache Maven
- LensKit
- A Java development environment, such as IntelliJ IDEA, Eclipse, or NetBeans
- You may find R useful (and it is used by the default LensKit project templates), but it will not be required for the coursework.

The remaining LensKit dependencies are either included in the LensKit download or automatically retrieved with Maven.

## A Note on Development Environments

You can use whatever Java development environment you wish to complete the coursework. It will be easiest if your environment supports Groovy, as the LensKit scripting language is built on Groovy.

We provide instructions for setting up IntelliJ IDEA and Eclipse. We will be using IntelliJ IDEA in the lecture videos. If you use a different development environment, such as NetBeans, Emacs, or Vim, you will be on your own. We do encourage you to make use of the forums to discuss how to configure various environments for completing the programming assignments.

## Setup Instructions

- [Windows](#)
- [Mac OS X](#)
- [Linux](#)

# Performance prediction and evaluation in Recommender Systems: an Information Retrieval perspective

Supervised by Prof. Pablo Castells and Dr. Iván Cantador.

Submitted in October 2012. Public defense in November 30, 2012.

---

[+]

## Abstract

Personalised recommender systems aim to help users access and retrieve relevant information or items from large collections, by automatically finding and suggesting products or services of likely interest based on observed evidence of the users' preferences. For many reasons, user preferences are difficult to guess, and therefore recommender systems have a considerable variance in their success ratio in estimating the user's tastes and interests. In such a scenario, self-predicting the chances that a recommendation is accurate before actually submitting it to a user becomes an interesting capability from many perspectives. Performance prediction has been studied in the context of search engines in the Information Retrieval field, but there is little if any prior research of this problem in the recommendation domain.

This thesis investigates the definition and formalisation of performance prediction methods for recommender systems. Specifically, we study adaptations of search performance predictors from the Information Retrieval field, and propose new predictors based on theories and models from Information Theory and Social Graph Theory. We show the instantiation of information-theoretical performance prediction methods on both rating and access log data, and the application of social-based predictors to social network structures.

Recommendation performance prediction is a relevant problem per se, because of its potential application to many uses. Thus, we primarily evaluate the quality of the proposed solutions in terms of the correlation between the predicted and the observed performance on test data. This assessment requires a clear recommender evaluation methodology against which the predictions can be contrasted. Given that the evaluation of recommender systems is an open area to a significant extent, the thesis addresses the evaluation methodology as a part of the researched problem. We analyse how the variations in the evaluation procedure may alter the apparent behaviour of performance predictors, and we propose approaches to avoid misleading observations.

In addition to the stand-alone assessment of the proposed predictors, we re-search the use of the predictive capability in the context of one of its common applications, namely the dynamic adjustment of hybrid methods combining several recommenders. We research approaches where the combination leans towards the algorithm that is predicted to perform best in each case, aiming to enhance the performance of the resulting hybrid configuration.

The thesis reports positive empirical evidence confirming both a significant predictive power for the proposed methods in different experiments, and consistent improvements in the performance of dynamic hybrid recommenders employing the proposed predictors.

# Document

You may download the [whole document](#), or each part separately:

- [Preface](#)
- [Part I](#) Introduction and Context
  - [Chapter 1](#) Introduction
  - [Chapter 2](#) Recommender systems
- [Part II](#) Evaluating performance in recommender systems
  - [Chapter 3](#) Evaluation of recommender systems
  - [Chapter 4](#) Ranking-based evaluation of recommender systems: experimental designs and biases
- [Part III](#) Predicting performance in recommender systems
  - [Chapter 5](#) Performance prediction in Information Retrieval
  - [Chapter 6](#) Performance prediction in recommender systems
- [Part IV](#) Applications
  - [Chapter 7](#) Dynamic recommender ensembles
  - [Chapter 8](#) Neighbour selection and weighting in user-based collaborative filtering
- [Part V](#) Conclusions
  - [Chapter 9](#) Conclusions and future work
- [Part VI](#) Appendices
  - [Appendix A](#) Materials and methods
  - [Appendix B](#) Introducción
  - [Appendix C](#) Conclusiones y trabajo futuro
- [References](#)

# Slides

Here you can find a PDF version of the [slides](#).

---

# Software

TODO

---

# Miscellanea

- Ficha [Teseo](#)
-

# Andriy Mnih

My email address can be easily derived from the URL for this page.

## About me

I am a research scientist at Google DeepMind. Until February 2013, I was a postdoctoral researcher at Gatsby, working with [Yee Whye Teh](#). Prior to that I was a PhD student in the [Machine Learning Group](#) at the University of Toronto, advised by [Geoffrey Hinton](#).

## Research interests

- latent variable models
- representation learning
- statistical language modelling

## Publications

### **Neural Variational Inference and Learning in Belief Networks**

Andriy Mnih and Karol Gregor

*International Conference on Machine Learning 2014 (ICML 2014)* [\[pdf\]](#) [\[slides\]](#) [\[poster\]](#) [\[bibtex\]](#)

### **Deep AutoRegressive Networks**

Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, Daan Wierstra

*International Conference on Machine Learning 2014 (ICML 2014)* [\[pdf\]](#) [\[bibtex\]](#)

### **Learning word embeddings efficiently with noise-contrastive estimation**

Andriy Mnih and Koray Kavukcuoglu

*Advances in Neural Information Processing Systems 26 (NIPS 2013)* [\[pdf\]](#) [\[poster\]](#) [\[bibtex\]](#)

### **Learning Label Trees for Probabilistic Modelling of Implicit Feedback**

Andriy Mnih and Yee Whye Teh

*Advances in Neural Information Processing Systems 25 (NIPS 2012)* [\[pdf\]](#) [\[poster\]](#) [\[bibtex\]](#)

### **A fast and simple algorithm for training neural probabilistic language models**

Andriy Mnih and Yee Whye Teh

*International Conference on Machine Learning 2012 (ICML 2012)* [\[pdf\]](#) [\[slides\]](#) [\[poster\]](#) [\[bibtex\]](#) [\[5 min talk\]](#)

### **Taxonomy-Informed Latent Factor Models for Implicit Feedback**

Andriy Mnih

*JMLR W&CP Volume 18: Proceedings of KDD Cup 2011* [\[pdf\]](#) [\[slides\]](#) [\[bibtex\]](#)

### **Learning Distributed Representations for Statistical Language Modelling and Collaborative Filtering**

Andriy Mnih

PhD Thesis, University of Toronto, 2009 [\[pdf\]](#) [\[bibtex\]](#)

### **Improving a Statistical Language Model Through Non-linear Prediction**

Andriy Mnih, Zhang Yuecheng, and Geoffrey Hinton

*Neurocomputing, 72:7-9, 2009* [[bibtex](#)]

### A Scalable Hierarchical Distributed Language Model

Andriy Mnih and Geoffrey Hinton

*Advances in Neural Information Processing Systems 21 (NIPS 2008)* [[pdf](#)] [[bibtex](#)]

### Bayesian Probabilistic Matrix Factorization using Markov Chain Monte Carlo

Ruslan Salakhutdinov and Andriy Mnih

*International Conference on Machine Learning 2008 (ICML 2008)* [[pdf](#)] [[bibtex](#)]

### Improving a Statistical Language Model by Modulating the Effects of Context Words

Zhang Yuecheng, Andriy Mnih, and Geoffrey Hinton

*European Symposium on Artificial Neural Networks 2008 (ESANN 2008)*

### Probabilistic Matrix Factorization

Ruslan Salakhutdinov and Andriy Mnih

*Advances in Neural Information Processing Systems 20 (NIPS 2007)* [[pdf](#)] [[bibtex](#)]

### Three New Graphical Models for Statistical Language Modelling

Andriy Mnih and Geoffrey Hinton

*International Conference on Machine Learning 2007 (ICML 2007)* [[pdf](#)] [[bibtex](#)]

### Restricted Boltzmann Machines for Collaborative Filtering

Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton

*International Conference on Machine Learning 2007 (ICML 2007)* [[pdf](#)] [[bibtex](#)]

### Visualizing Similarity Data with a Mixture of Maps

James Cook, Ilya Sutskever, Andriy Mnih, and Geoffrey Hinton

*AI and Statistics 2007 (AISTATS 2007)* [[pdf](#)] [[bibtex](#)]

### Learning Nonlinear Constraints with Contrastive Backpropagation

Andriy Mnih and Geoffrey Hinton

*International Joint Conference on Neural Networks 2005 (IJCNN 2005)* [[bibtex](#)]

### Wormholes Improve Contrastive Divergence

Geoffrey Hinton, Max Welling, and Andriy Mnih

*Advances in Neural Information Processing Systems 16 (NIPS 2003)* [[bibtex](#)]

## Explaining the user experience of recommender systems

Bart P. Knijnenburg · Martijn C. Willemsen ·  
Zeno Gantner · Hakan Soncu · Chris Newell

Received: 30 November 2010 / Accepted in revised form: 30 August 2011 /

Published online: 10 March 2012

© The Author(s) 2012. This article is published with open access at Springerlink.com

**Abstract** Research on recommender systems typically focuses on the accuracy of prediction algorithms. Because accuracy only partially constitutes the user experience of a recommender system, this paper proposes a framework that takes a user-centric approach to recommender system evaluation. The framework links objective system aspects to objective user behavior through a series of perceptual and evaluative constructs (called subjective system aspects and experience, respectively). Furthermore, it incorporates the influence of personal and situational characteristics on the user experience. This paper reviews how current literature maps to the framework and identifies several gaps in existing work. Consequently, the framework is validated

---

B. P. Knijnenburg (✉)

Department of Informatics, Donald Bren School of Information and Computer Sciences,  
University of California, Irvine, CA 92697, USA  
e-mail: bart.k@uci.edu

B. P. Knijnenburg · M. C. Willemsen

Human-Technology Interaction Group, School of Innovation Sciences, Eindhoven University  
of Technology (TU/e), P.O. Box 513, 5600 MB, Eindhoven, The Netherlands  
e-mail: m.c.willemsen@tue.nl

Z. Gantner

Information Systems and Machine Learning Lab (ISMLL), University of Hildesheim,  
Marienburger Platz 22, 31141 Hildesheim, Germany  
e-mail: gantner@isml.de

H. Soncu

European Microsoft Innovation Center GmbH, Ritterstrasse 23, 52072 Aachen, Germany  
e-mail: hakan.soncu@microsoft.com

C. Newell

BBC Research & Development, Centre House, 56 Wood Lane, London, W12 7SB, UK  
e-mail: chris.newell@rd.bbc.co.uk

with four field trials and two controlled experiments and analyzed using Structural Equation Modeling. The results of these studies show that subjective system aspects and experience variables are invaluable in *explaining why and how the user experience of recommender systems comes about*. In all studies we observe that perceptions of recommendation quality and/or variety are important mediators in predicting the effects of objective system aspects on the three components of user experience: process (e.g. perceived effort, difficulty), system (e.g. perceived system effectiveness) and outcome (e.g. choice satisfaction). Furthermore, we find that these subjective aspects have strong and sometimes interesting behavioral correlates (e.g. reduced browsing indicates higher system effectiveness). They also show several tradeoffs between system aspects and personal and situational characteristics (e.g. the amount of preference feedback users provide is a tradeoff between perceived system usefulness and privacy concerns). These results, as well as the validated framework itself, provide a platform for future research on the user-centric evaluation of recommender systems.

**Keywords** Recommender systems · Decision support systems · User experience · User-centric evaluation · Decision-making · Human-computer interaction · User testing · Preference elicitation · Privacy

## 1 Introduction

Recommender systems are designed to help the user make better choices from large content catalogs, containing items as distinct as books, movies, laptops, cameras, jokes, and insurance policies (Xiao and Benbasat 2007; Resnick and Varian 1997). Before the advent of recommender systems, such content-based systems would offer users the entire catalog (possibly with a generic search/filter feature). Recommender systems, on the other hand, offer each user a personalized subset of items, tailored to the user's preferences. The system derives these user preferences from implicit or explicit feedback (Pommeranz et al. 2012). Implicit feedback recommenders analyze clicking/purchasing behavior (e.g. amazon.com, see also Hauser et al. 2009). Explicit feedback recommenders let users rate items, (e.g. youtube.com, see also McNee et al. 2002; Cena et al. 2010; Gena et al. 2011), critique items (see also Chen and Pu 2012; Viappiani et al. 2006), assign weights to item attributes (see also Häubl et al. 2004), or indicate their specific needs (e.g. HP.com 'help me choose', see also Felix et al. 2001). Finally, the system calculates recommendations by comparing the user's preferences to the features of the catalog items (content-based recommender systems), or to other users' preferences (collaborative filtering recommenders).

A typical interaction proceeds as follows: First, the user's preferences are elicited. Based on the collected preference data, the system tries to predict how much the user would appreciate each of the available items in the catalog. Finally, the system presents the user those items that have the highest predicted value to the user. In some recommender systems this terminates the interaction, in other systems the users continue to indicate their preferences and receive recommendations continually.

An essential aspect of any recommender system is the algorithm that provides personalized recommendations based on the user's preferences (Burke 2002). The more

accurate the predictions of this algorithm, the more accurately the system can predict the best recommendations for the user. Not surprisingly, a significant part of the research on recommender systems concerns creating and evaluating better prediction algorithms (McNee et al. 2006a; Cosley et al. 2003; Ziegler et al. 2005). An excellent overview of available algorithms can be found in Burke (2002) and in Adomavicius and Tuzhilin (2005); more recent approaches were presented in Koren et al. (2009), Koren (2010) and Hu et al. (2008). Herlocker et al. (2004) provide a thorough discussion of available evaluation metrics.

The premise of this algorithm research is that better algorithms lead to perceptibly better recommendations, which in turn lead to better user experience in terms of choice satisfaction and perceived system effectiveness. However, several researchers have argued that there are other factors that influence the *user experience* (users' subjective evaluation of their interaction with the system), and that these factors have not received the amount of attention they deserve (McNee et al. 2006a,b; Cosley et al. 2003; Murray and Häubl 2008, 2009; Ozok et al. 2010; Pu et al. 2012; Konstan and Riedl 2012). System aspects other than accuracy can influence satisfaction and other evaluative measures (e.g. diversification; Ziegler et al. 2005; Willemse et al. 2011). Furthermore, situational or personal aspects (e.g. product expertise; Kamis and Davern 2004; Knijnenburg and Willemsen 2009, 2010; Knijnenburg et al. 2011; and privacy concerns; Teltzrow and Kobsa 2004; Komiak and Benbasat 2006) can also influence how people interact with and evaluate the system. Unfortunately, even studies that consider aspects other than accuracy look at a limited set of variables that influence each other (e.g., how satisfaction changes due to a diversification, or how choices become more accurate with the inclusion of a recommender engine) without integrating these variables into a model of overall user experience.

An integrated view on the user experience of recommender systems can be obtained by means of user-centric development (McNee et al. 2006b) and evaluation (Pu and Chen 2010; Pu et al. 2012). The current paper therefore extends and tests our user-centric evaluation framework for recommender systems proposed in Knijnenburg et al. (2010a). To understand and improve the user experience of recommender systems, it is necessary to conduct empirical evaluations that consider the entire process of how the user experience comes about. Therefore, our framework describes how objective aspects of the system (e.g. the algorithms used) are subjectively perceived by the user (e.g. if they perceive differences in recommendation quality for these different algorithms), and how these perceptions, together with personal and situational characteristics, result in specific user experience and interaction with the system (e.g. whether a higher perceived recommendation quality leads to a more positive evaluation of the system, a higher satisfaction with the chosen items, and a change in user behavior). Such a framework will provide a deeper understanding of how objective system aspects influence the user experience and behavior through perceived system aspects. It thereby allows for a better understanding of *why and how* certain aspects of the system result in a better user experience and others do not, which helps further user-centric research and development of recommender systems.

## 2 Components of the framework

The main goal of our framework is to provide a set of structurally related concepts that can be used in empirical studies to describe and measure the user experience of recommender systems. User experience is an ill-defined concept, and lacks well-developed assessment methods and metrics (McNamara and Kirakowski 2006; Law et al. 2009). In our framework, we distinguish between objective system aspects, (e.g. algorithms, user interface features), subjective system aspects (users' perceptions of these objective system aspects), and user experience (users' evaluations of their interaction with the system) and interaction (users' behaviors). We also consider the context of the interaction in terms of personal and situational characteristics. Before we describe the framework itself, we will discuss several theories that served as a basis for our framework.

### 2.1 Existing theories

#### 2.1.1 Normative and attitudinal models

At the core of many psychological models of human behavior is the Theory of Reasoned Action (TRA) by Fishbein and Ajzen (1975). This theory claims that attitudinal and normative factors influence behavioral intention, which in turn predicts actual behavior. Davis et al. (1989, see also Davis 1989) adopted the attitudinal part of this theory in their Technology Acceptance Model (TAM). In the TAM, the attitude towards using a technology is explained by the perceived usefulness and perceived ease of use of the system. Venkatesh et al. (2003) created a similar theory called the Unified Theory of Acceptance and Use of Technology (UTAUT), based on the normative part of TRA, showing how personal and situational characteristics can influence behavioral intention. In the UTAUT, attitudinal concepts are entirely replaced by more experience-related evaluative concepts (performance expectancy, effort expectancy, social influence, and facilitating conditions).

With respect to our framework, these theories make a distinction between behaviors (and behavioral intentions) and the attitudes that cause these behaviors. These attitudes are in turn caused by experiential factors like perceived usefulness and ease of use (TAM), and by personal and situational characteristics (UTAUT).

#### 2.1.2 User experience models

Hassenzahl (2008) defines user experience (UX) as “a momentary, primarily evaluative feeling (good-bad) while interacting with a product or service. Good UX is the consequence of fulfilling the human needs for autonomy, competence, stimulation (self-oriented) through interacting with the product or service (i.e. hedonic quality).” Hassenzahl’s (2005) model of user experience describes how certain objective aspects of the system (e.g. its interaction and presentation style) are perceived in terms of pragmatic attributes (i.e. does the system deliver high quality results in an effortless

way?) and hedonic attributes (i.e. does it stimulate, is it desirable?). These perceptions in turn cause an experiential evaluation in terms of appeal, pleasure and satisfaction.

With respect to our framework, Hassenzahl's model links objective system aspects to evaluative experiential factors through subjective perceptions. The distinction between perception and evaluation is subtle but important: Perception denotes whether certain objective system aspects register with the user at all, while evaluation denotes whether the perceived aspect has any personal relevance to the user. This may for instance give us an insight in why users may perceive a change in recommendation quality but at the same time do not show a change in experience or behavior.

Furthermore, whereas the TRA-related theories are restricted to pragmatic attributes, Hassenzahl also stresses the importance of hedonic attributes. Experimental evidence shows that hedonic attributes like pleasure and 'flow' (a feeling of automatic and highly focused interaction; [Csikszentmihalyi 1975](#)) indeed also determine the user experience ([Koufaris 2003](#); [Hsu and Lu 2004](#); [Yu et al. 2005](#)).

### 2.1.3 User experience models for recommender systems

[Hayes et al. \(2002\)](#) propose a framework for testing the user satisfaction of recommender algorithms in operational systems. Their approach is restricted to behavioral measures of satisfaction, and their focus is primarily on the algorithm. Furthermore, Hayes et al.'s work has limits because they advocate a setup in which several algorithms are tested at the same time for the same user, an approach which provides a significant departure from the normal user experience of a recommender system which generally employs only one algorithm at a time.

[Zins and Bauernfeind \(2005\)](#) constructed a model of the user experience of recommender systems based on a survey conducted among users of two travel recommenders and a system for finding digital cameras. Their model shows how personal characteristics influence trust, flow, and browsing behavior, and how these in turn influence system satisfaction. A clear limitation of their model is that it does not explain how objective system aspects may influence the user experience.

[McNee et al. \(2006b\)](#) created an analytic model of Human-Recommender Interaction (HRI) for the development of recommender systems. The goals and tasks of the users are analyzed and used to determine the appropriate recommender system dialogue and 'personality'. McNee et al.'s model clearly serves a different purpose than our framework (development as opposed to evaluation). They do however suggest linking subjective HRI metrics to traditional objective performance metrics of algorithm accuracy, and stress the importance of the context (e.g. users' goals and tasks) in which recommendations are made.

[Xiao and Benbasat \(2007\)](#) presented an extensive literature review of the marketing-oriented research on recommender systems. Their overview, too, provides insight into the mechanisms underlying the user experience of recommender systems, albeit from many different studies (each focusing just on one part of the entire experience). Their resulting framework shows how certain characteristics of recommender systems cause changes in users' evaluation and decision-making behaviors, and in their adoption of the recommender system. It also includes personal and in situational characteristics that moderate these effects. The framework we present below bears a lot of similarity

to Xiao and Benbasat's (2007) framework, but goes beyond it by including subjective system aspects. Moreover, while their framework is constructed mainly for the purpose of summarizing existing research, we pose our framework as a starting-point for the evaluation of recommender systems.

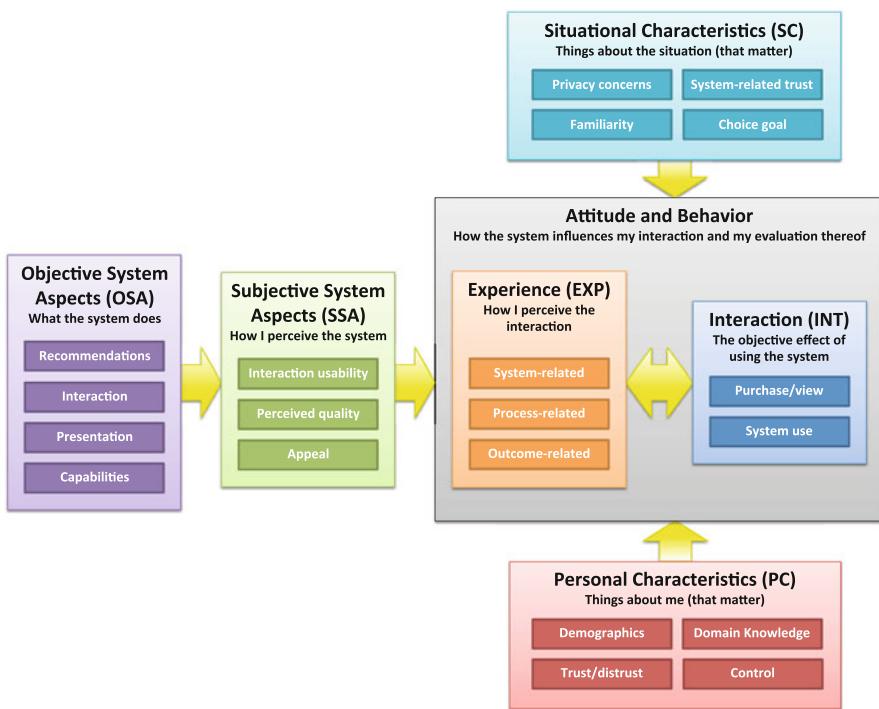
Pu and Chen (2010, see also Pu et al. 2012) provide an extensive questionnaire to test several specific experience concepts of recommender systems. Their research model also explicitly considers perceived system qualities as antecedents of user beliefs, attitudes and behavioral intentions, and in that way it is similar to our framework. Our framework, however, takes a more abstract approach by providing a description of the structural relationships between the general, higher level concepts that play a role in user experience, without strictly specifying operational, lower level constructs and the questions that measure them.<sup>1</sup> To answer specific research questions, researchers need to define and operationalize a set of specific, lower level constructs, and Pu and Chen's questionnaires can be used as a starting point for this operationalization; another option is to use the pragmatic procedure for recommender system evaluation that is based on our framework (Knijnenburg et al. 2011a). However, since user experience is highly contingent upon the purpose of the system under evaluation, the specific concepts and specific questionnaire items to measure these concepts may differ from study to study. Moreover, Pu and Chen's model does not include context (personal and situational characteristics), and it does not suggest how objective system aspects influence the various constructs in their framework, making it more difficult to select concepts from their framework for a particular user study.

Ozok et al. (2010) provide a wide range of design guidelines based on a questionnaire of recommender system usability. Their results describe the effects of specific system aspects on the usability of recommender systems. However, they employ a descriptive approach, which relies on the users' stated opinions about recommender systems in general instead of experimental manipulations of a specific system.

## 2.2 The main structure of the framework explained

Figure 1 shows our framework. Like Hassenzahl (2005) and Xiao and Benbasat (2007), we take objective system aspects (OSA) as a starting point for the evaluation. The objective system aspects consist of the algorithms used by the system, the visual and interaction design of the system, the way it presents the recommendations, and additional features such as social networking. Like Hassenzahl (2005), we link these objective system aspects to subjective system aspects (SSA), which represent users' perception of the objective system aspects. Also like in Hassenzahl (2005) model, these subjective system aspects include both pragmatic characteristics (usability and quality) and hedonic characteristics (appeal). The subjective system aspects, measured with questionnaires, are expected to mediate the influence of the objective system aspects on the user experience. The main reason for including subjective system aspects as mediators is that recommender systems provide a personalized experience

<sup>1</sup> In this respect, our framework resembles Fishbein and Ajzen (1975) TRA, which also instructs researchers to elicit a different set of personal and normative beliefs to measure per experiment.



**Fig. 1** Our framework for the user-centric evaluation of recommender systems

to the user, and that this personalization may not be equally apparent to all users. The subjective system aspects can show whether the objective aspects are perceived at all.

Like all existing models, our framework carefully distinguishes between attitude and behavior,<sup>2</sup> although we use the more specific terms experience and interaction. The *experience* (EXP) signifies users' evaluation of the system. In that way it is closely related to attitudes in TAM (or rather the factors that cause them), with the addition of hedonic aspects (like in Hassenzahl's (2005) model). Experience is also measured with questionnaires, and is conceptually divided into the evaluation of the system (system-EXP), the evaluation of the decision process (process-EXP), and the evaluation of the final decisions made (outcome-EXP). The *interaction* (INT) is the observable behavior of the user. A complex interplay exists between interaction and experience: a positive user experience changes the interaction, but the interaction is also what initially caused the user experience.

Like the models of Xiao and Benbasat (2007) and Venkatesh et al. (2003), our model asserts that experience and interaction typically also depend on *personal* and *situational characteristics* (referred to as PC and SC). Personal characteristics include demographics, trust, domain knowledge, and perceived control (the latter two are

<sup>2</sup> To emphasize this distinction we have slightly altered the labels in the framework since our previous publications.

prominent in TRA). Situational characteristics are dependent on the context of the interaction; at different points in time, users may have different choice goals, trust and privacy concerns, and familiarity with the system (McNee et al. 2006b).

### 3 Expected benefits of our framework

Our framework explicitly links the objective interaction (INT) to objective system aspects (OSA) through a series of subjective constructs (SSA and EXP). The framework can be used as a guideline for controlled user experience tests. Specifically, by manipulating a certain system aspect (OSA) in a controlled experiment (keeping all other aspects the same), one can identify the effect of this aspect on the users' perceptions (SSA), experience (EXP) and behaviors (INT). By careful manipulation of specific objective system aspects, researchers can uncover generic truths about recommender systems. These can then inform the design and development of future versions of the studied aspects.

Moreover, the framework allows one to conduct empirical evaluations in a more integrative fashion than most existing recommender systems research: It allows researchers to consider the interplay between multiple objective system aspects (e.g. algorithm versus interface), as well as to investigate the trade-offs between several aspects of user experience (e.g. satisfaction versus choice difficulty).

The framework provides insight into the relationships between the general concepts that play a role in the user experience of recommender systems. By tailoring the operationalization of these general concepts to the specific system under evaluation, the framework can be applied to a range of different types of consumer-facing recommender systems, including e-commerce recommenders (recommending products), media recommenders (recommending, for example videos, music or news articles) and social network recommenders (recommending users to befriend or follow). To exemplify this point, in the remainder of this section we use our framework to map a wide array of existing research. Most of this existing research (as well as our own empirical work) specifically considers what in the realm of e-commerce recommenders has been called "experience products", for which the quality is hard to determine before purchase, as opposed to "search products", which have attributes that can be considered before the decision is made (Nelson 1970). Because it is hard to determine the quality of experience items before the actual decision is made, decision-making processes for such items typically rely more heavily on recommendations and other external sources of information (Bhatnagar and Ghose 2004; Stolze and Nart 2004; Huang et al. 2009; Ochi et al. 2010).

To conclude this section, we indicate specific gaps in current knowledge, and provide an overview of the research opportunities that these gaps present. In the subsequent section, we present the results of several *empirical evaluations* that use parts of the framework for their main hypotheses. After presenting the results of these evaluations one by one, the findings will be integrated under the generic concepts of the framework. This will allow us to validate the framework, and to take a first step towards bridging the uncovered gaps. To effectively integrate our findings, we limited our empirical evaluations to media recommenders.

### 3.1 The objects of user experience evaluation

User experience as defined in our framework (EXP) is not a one-dimensional concept; it may entail various aspects (broadly ranging from pragmatic to hedonic concepts) and several different *objects of evaluation*. Especially for recommender systems, knowing the object of evaluation is critical in understanding the dynamics of the user experience (Pathak et al. 2010; Tintarev and Masthoff 2012): When we say that the user experience of recommender system X is better than that of system Y, are we evaluating the system, the process of using the system to get to a decision, or the chosen item itself? This distinction is important, as different system aspects may influence different objects of user experience; a visually attractive interface may improve the evaluation of the system (system-EXP), a good preference elicitation method may make decisions easier (process-EXP), and an accurate algorithm may increase the quality of the final decision (outcome-EXP).

Furthermore, the evaluations of the different experience objects may influence each other. For instance, a positive evaluation of the chosen item(s) may “rub off” on the evaluation of the system. To capture the multi-faceted nature of user-experience, our framework therefore considers each of these objects: the system, the process, and the outcome.

In current research, however, this is rarely done; researchers in different domains use different objects of evaluation. Particularly, marketing and decision-making researchers mainly look at the outcome-EXP variables such as the quality of the choice, the users’ confidence in making the right choice, and the satisfaction with the chosen item (Hostler et al. 2005; Pedersen 2000; Vijayasarathy and Jones 2001; Krishnan et al. 2008; Bechwati and Xia 2003). They rarely take the system or the choice process as focal points of evaluation.

Human–computer interaction (HCI) researchers have traditionally been more comprehensive in the coverage of all objects of user experience of their research. However, this field has a tendency towards formative evaluations such as Think Aloud testing and Heuristic Evaluation (Van Velsen et al. 2008). The results of such evaluations are limited in generalizability, and therefore not the focus of this paper. The available summative evaluations in the HCI field primarily report on system-EXP variables such as system quality and user loyalty (also operationalized as the intention to return), on process-EXP variables such as cognitive effort and competence in using the system, and on outcome-EXP variables such as decision satisfaction (Pu and Chen 2007; Chen and Pu 2009; Pu et al. 2008; Bharati and Chaudhury 2004; Ochi et al. 2010; Hu and Pu 2009, 2011; Jones et al. 2009; Felfernig et al. 2007).

According to our framework (Fig. 1), these user experience effects do not stand alone, but are instead part of a larger chain of effects. Specifically, in the following sections we will argue that the user experience (EXP) is caused by objective system aspects (OSA, via SSA) and personal or situational characteristics (PC or SC). Moreover, the experience variables themselves may be structurally related to one another: Bharati and Chaudhury (2004), for instance, showed that the perceived quality of the system (system-EXP) positively influences the decision satisfaction (outcome-EXP). Although Bharati and Chaudhury investigate the objective system aspects in their study

(by manipulating the recommender system under evaluation), they do not include the objective system aspects in the analyzed chain of effects.

### 3.2 From accuracy to user experience

A large part of existing recommender systems research is focused on creating better prediction algorithms, thereby implicitly assuming that better algorithms will lead to a better user experience. Explicitly testing this assumption would require empirical evaluations with real users on real systems. Several researchers in marketing and decision-making conducted such user-centric evaluations of their recommender systems. For instance, they looked at the reduction in choice effort through a recommender system (Häubl et al. 2004; Pedersen 2000; Vijayasarathy and Jones 2001; Diehl et al. 2003; Häubl and Trifts 2000; Hostler et al. 2005). However, they usually compare a recommender system against a system without recommendation features (or a recommender system against no system at all), rather than looking at the often subtle differences between algorithms. The results of such unbalanced comparisons, in which the “personalized” condition clearly has an advantage over the non-personalized condition, are usually unsurprising (see Van Velsen et al. 2008). However, Chin (2001) argues that this advantage is not always apparent and that a comparison with a non-personalized system may very well be justified. Some researchers compare a recommender system against human recommenders (Krishnan et al. 2008; Stolze and Nart 2004), but these studies provide little insight into the effect of algorithm accuracy on the user experience; for that, several algorithms should be pitted against each other.

Surprisingly few studies compare algorithms in live experiments with real users. Researchers who do compare the user experience effects of several algorithms find surprising results. In a comparison of six recommender algorithms, McNee et al. (2002) found that although the “Item-Item CF” algorithm provided the best predictions, users rated it the least helpful. Torres et al. (2004) found that although the “CBF-separated CF” approach had the lowest predictive accuracy among five algorithms, this approach resulted in the highest user satisfaction. In other words, the presumed link between algorithm accuracy (an OSA) and user experience (EXP) is all but evident. Our framework allows researchers of recommender systems to take a step beyond algorithmic accuracy (OSA) towards its effects on user experience (EXP).

### 3.3 Subjective system aspects as mediators

The presented framework indicates that the apparent missing link between algorithm accuracy and user experience can be found in mediation through perception. The link between algorithm accuracy (an OSA) and user experience (EXP) is often weak (Chin 2001), and can then only be established by including the mediation through the users’ perception of the algorithm accuracy (an SSA). In other words, the framework hypothesizes that users can perceive algorithm accuracy, and that this perception influences the experience (OSA → SSA → EXP).

In light of these hypotheses, existing research has established that users are, in several instances, able to observe objective differences in recommendation quality (in

terms of the framework: OSA → SSA; for examples, see [Ziegler et al. 2005](#); [Cosley et al. 2003](#)). It is however not clear how these (typically subtle) differences in perceived recommendation quality affect the user experience (SSA → EXP), because few researchers have tested the effect of their algorithms on the users' perception, behavior *and* experience.

This gap in existing research (i.e. not measuring the SSA as a mediator between OSA and EXP) makes it hard to explain why in some experiments better algorithms do not lead to a better experience. One possible reason might be that users were not able to notice the quality differences (the OSA does not affect the SSA), e.g., the quality differences between two algorithms may have been too small to notice, and thus would not influence the user experience. Another possible explanation (which is not mutually exclusive) might be that users may have observed the quality differences, but may just not have been influenced by these differences in their experience (no link between SSA and EXP), e.g., they may actually like to see good recommendations accompanied with some bad ones, as it makes their final decision easier to justify. Finally, an effect of accuracy on experience may exist, but just be overshadowed by individual differences of perception (this is not unlikely for recommender systems, as their effect is not equally pronounced for each user). In such case one should measure whether the user actually noticed the quality difference or not (SSA is needed as a mediator between OSA and EXP).

The inclusion of SSAs may thus increase the *robustness* of the effects of the OSAs on EXP. Moreover, SSAs provide a more thorough understanding of *how and why* certain features of a recommender system affect the user experience. This does not only hold for algorithm accuracy, but for any manipulated objective system aspects. [Chen and Pu \(2009\)](#) have created a chain of effects from objective algorithm accuracy (OSA) to user-perceived algorithm accuracy (SSA), from objective user effort (OSA) to user-perceived effort (SSA), and from the perceived accuracy and effort (SSA) to intention to purchase and intention to return (INT). They find significant differences between their two tested interfaces for each of these constructs. This path analysis is an important step towards an integrated analysis of recommender system experience, but in our approach we extend it in two directions: First of all, we include the manipulations that cause the objective differences in the model. In the Chen and Pu study they cause significant differences in each construct individually, but an inclusion of the manipulation as a dummy variable into the path model would allow for a mediation analysis of the experimental effects. Secondly, we add constructs explicitly asking the users about their experience (EXP).

### 3.4 Triangulation of (logged) behavioral data

Although user experience (EXP) is mainly a subjective phenomenon, its effects will likely be reflected in the users' observable behavior (INT). This idea is a fundamental property of all theories based on [Fishbein and Ajzen's \(1975\)](#) Theory of Reasoned Action, although we do not take the direction of the effect to be merely one-way. Whereas attitude causes behavior in TRA, our focus on experience (which is a much more interactive concept than attitude) also considers the inverse effect. For example:

users who are more satisfied may increase their usage of the system ( $\text{EXP} \rightarrow \text{INT}$ ), while at the same time increased usage may cause an increase in satisfaction ( $\text{INT} \rightarrow \text{EXP}$ ).

Researchers in the field of algorithm accuracy predominantly use behavioral data for their evaluations: they use logged clicks (either item selections or ratings) to train and test their algorithms (Konstan and Riedl 2012). Researchers in marketing and decision-making also analyze behavioral data, but focus more on decision time, switching behavior after the choice, or total consumption volume (Häubl et al. 2004; Pedersen 2000; Vijayasarathy and Jones 2001; Stolze and Nart 2004; Hostler et al. 2005; Ho and Tam 2005; Tam and Ho 2005; Pathak et al. 2010). A common problem with behavioral data, however, is that they are not always good indicators of users' subjective experience. For instance, Pu and Chen (2006) found that the actual time users spent looking at items in their system did not correlate with users' subjective perceptions, And Spiekermann et al. (2001) found that stated willingness to provide feedback did not correlate with actual feedback behavior.

Another problem with behavioral data is that their interpretation is often troublesome (Van Velsen et al. 2008). For instance, if users stay on a video clip recommendation site for a longer time, does this mean that the efficiency of the system is low (it takes longer for users to find what they want), or that the users enjoy the site more (to the point that they stay longer to watch more clips)? To solve this dilemma, Van Velsen et al. (2008) suggests to "triangulate" the objective behavioral data (INT) with the subjective experience data (EXP) gathered through other methods (e.g. questionnaires).

From a commercial perspective, influencing the users' objective behavior may seem to be the primary objective of recommender systems research, such as getting the user to buy more products (in e-commerce recommenders) or watch more advertisements (in media recommenders). However, experience concepts reflect and influence users' attitudes towards a system, and research shows that positive attitudes are related to increased adoption rates (Fishbein and Ajzen 1975; Davis et al. 1989; Venkatesh et al. 2003). To get an indication of the longer-term effects of the system, behavioral data should thus be complemented with subjective experience measurements. In our framework, behavioral data is therefore correlated (triangulated) with subjectively measured experience concepts.

### 3.5 Personal and situational characteristics in context

The user experience cannot be entirely attributed to the recommender system itself, it may also depend on characteristics of the user (Personal Characteristics, or PC) and the situation in which the user is using the system (Situational Characteristics, or SC) (Chin 2001). These factors are typically beyond the influence of the recommender system, but do influence the user experience.

Domain knowledge (or 'expertise') is an important PC variable in this respect: Kamis and Davern (2004) show that users with a higher level of domain knowledge perceive recommender systems as less useful and harder to use than novices. In a music recommender experiment, Hu and Pu (2010) show that expert users perceive

recommendations as less accurate, and the system as less helpful. They also state that they would use the system less. Overall, users with a moderate level of expertise rate the system as most effective.

Users' trust in the system may also influence their experience, and vice versa. [Komiak and Benbasat \(2006\)](#) show that good recommendations can increase trust in both the competence and the integrity of a recommender system, and that a higher level of trust eventually leads to an increased intention to adopt the system. [Wang and Benbasat \(2007\)](#) show that trust in recommender systems is furthermore caused by disposition (the user's initial level of trust), calculation (the user's estimation of the costs and benefits for the system to be trustworthy), interaction (the user's expectations about the system, control over the system, and validation of the system results) and knowledge (an inference based on what the user knows about the system). This makes trust both a PC (depending on the user's personality) and an SC variable (depending on the user, the system and the situation).

Very few studies have investigated which personal and situational characteristics exactly motivate and inhibit users to provide preference feedback to the system (see [Pommeranz et al. 2012](#), for a notable exception). This is an important issue, as many recommender systems rely on explicit feedback (e.g. users' ratings) to give good recommendations. Privacy concerns may reduce users' tendency to disclose personal information ([Teltzrow and Kobsa 2004; Chellappa and Sin 2005; Berendt and Teltzrow 2005; Ackerman et al. 1999](#)). On the other hand, if it positively influences their user experience (i.e. in terms of better recommendations), users may be more willing to provide feedback ([Spiekermann et al. 2001; Brodie et al. 2004; Kobsa and Teltzrow 2005](#)).

The main shortcoming of existing research on personal and situational characteristics is that these characteristics are often investigated in isolation (again, see [Pommeranz et al. 2012](#), for a notable exception). This makes it hard to evaluate the impact of these characteristics on the user experience relative to other possible factors that influence the user experience (e.g. is a certain PC → EXP more substantive than a certain SSA → EXP?), and to prove the effectiveness of possible remedies for negative influences (e.g. can a certain positive SSA → EXP offset a certain negative PC → EXP?).

In our framework personal and situational characteristics influence the user experience, but we explicitly describe such effects in addition to the effects of manipulated system aspects. This allows for judgments of relative importance, which are investigated thoroughly in our empirical evaluations.

### 3.6 Integration of user interface research

Both industry practitioners and academic researchers have argued that the interface of a recommender system may have far larger effects on users' experience with the recommender than the recommender's algorithmic performance ([McNee et al. 2006a; Baudisch and Terveen 1999; Murray and Häubl 2008; Xiao and Benbasat 2007; Ziegler et al. 2005; Ozok et al. 2010](#)). Below we provide a brief overview of user interface aspects (OSAs) influencing the user experience (EXP) of recommender systems.

### 3.6.1 Preference elicitation method

The preference elicitation method is the way in which the recommender system discovers what the user likes and dislikes. In content-based recommender systems, users may indicate their preference by assigning weights to attributes (Häubl et al. 2004; Kramer 2007), prioritizing user needs (Felix et al. 2001; Stolze and Nart 2004; Hu and Pu 2009) or critiquing examples (Pu and Chen 2006; Pu et al. 2008; Chen and Pu 2012; Viappiani et al. 2006, 2008). The research on these different preference elicitation methods shows that they have a substantive impact on the user experience (Chen and Pu 2012). Moreover, the optimal preference elicitation method may depend on user characteristics such as domain knowledge (Knijnenburg and Willemsen 2009, 2010; Knijnenburg et al. 2011).

In collaborative filtering recommender systems, the two most common preference elicitation methods are explicit and implicit elicitation. In explicit elicitation, users rate the items with, for example, one to five stars (see Gena et al. 2011, and Pommeranz et al. 2012, for a user-centric exploration of various alternative explicit elicitation methods). In implicit elicitation, preferences are derived from an analysis of the browsing and selection behavior of users. Research shows that a combination of explicit and implicit elicitation results in a higher recommendation accuracy (Koren et al. 2009), but no research has investigated differences in user experience and behavior between explicit and implicit elicitation.<sup>3</sup> Our framework provides the opportunity to investigate the effects of preference elicitation beyond accuracy.

### 3.6.2 Size and composition of recommendation sets

Most recommender systems provide an ordered list of recommendations. Whereas a substantial amount of research considers the individual qualities of these recommendations, little research has considered the composition of the list (Hu and Pu 2011; Chen and Pu 2012; Ziegler et al. 2005; Cooke et al. 2002). The composition may play an important role in the user experience of a recommender system, because it influences the users' decision-making process through context effects (Simonson and Tversky 1992; Tam and Ho 2005).

It is also unclear how many recommendations the system should provide. In conventional choice situations, too few items may restrict the users' freedom of choice, whereas too many items may lead to choice overload (a process-EXP variable, Schwartz 2004; Iyengar and Lepper 2000; Scheibehenne et al. 2010). In the context of recommender systems, where all recommended items are highly relevant, this choice overload effect may be even more prominent. When embedded in a user interface, a longer list of recommendations may enjoy the added benefit of attracting more attention (Tam and Ho 2005).

<sup>3</sup> Algorithms are often designed to handle a specific type of data (e.g. binary, five star rating) and are therefore restricted to a specific preference elicitation method. In practice, they are therefore often treated as one and the same thing. However, to get a more nuanced understanding of the specific effects of algorithms and preference elicitation methods, we make an explicit distinction between these two system aspects.

The order in which recommendations are presented also seems to have an effect on the users' experience and interaction. The consequences of such serial positioning effects are however unclear: [Lynch and Ariely \(2000\)](#) find that sorting by quality reduces price sensitivity in an e-commerce recommender; [Diehl et al. \(2003\)](#), however, find that it increases price sensitivity. [Tam and Ho \(2005\)](#) find that making one recommendation stand out increases user attraction, elaboration, and choice likelihood of that recommendation.

[Hu and Pu \(2011, see also Chen and Pu 2012\)](#) show that putting a logical structure on the list of recommendations (specifically, categorizing the recommendations to reflect different trade-offs) leads to a higher perceived categorical diversity of the recommendations, a higher satisfaction and decision confidence, and a higher intention to reuse the system and purchase items with it.

[Ziegler et al. \(2005\)](#) found that, up to a certain point, sacrificing (actual and perceived) individual recommendation quality in favor of recommendation set diversity can lead to a more positive subjective evaluation of the recommendation set (SSA; see also [Bradley and Smyth 2001](#)). This finding should be compared with other factors influencing the user experience to verify the robustness and extent of this effect. For instance, [Willemse et al. \(2011\)](#) find that diversification may reduce the choice difficulty (SSA), which further improves the user experience (EXP). Our framework provides a starting point for investigating the impact of the size and composition of recommendation sets on the user experience.

Finally, one can also think about *when* to provide recommendations. [Ho and Tam \(2005\)](#) find that users are most susceptible to be persuaded by recommendations in the earlier stages of the decision process.

### 3.6.3 Explanations

Another part of the presentation of recommendations is the possibility to explain why certain items are recommended. [Herlocker et al. \(2000\)](#) found that users like explanations in collaborative filtering recommender systems. Studying knowledge-based recommenders, [Felfernig et al. \(2007\)](#) and [Cramer et al. \(2008a,b\)](#) show that explanations increase the users' perception of the system's competence (system-EXP) and their trust in the quality of the recommendations (SSA). [Tintarev and Masthoff \(2012\)](#) show that users tend to like personalized explanations (i.e. explanations that highlight facts related to their preferences), but that these may actually be less effective than generic explanations.

## 3.7 Research opportunities

It is important to realize that our framework is not merely a classification of the important aspects of the user experience of recommender systems. Nor is it a pre-defined metric for standardized “performance” tests. Instead, it is a generic guideline for in-depth empirical research on the user experience of recommender systems; it conceptually defines a generic chain of effects that helps researchers to *explain why and how* the user experience of recommender systems comes about. This explanation

is the main value of the user-centric evaluation of recommender systems (McNee et al. 2006b).

The remainder of this paper will describe the empirical evaluations of our own recommender systems: a number of studies that together comprise a preliminary validation of parts of the evaluation framework. In these studies we have tried to repeatedly test a limited set of core variables of our framework. Additionally, the reviewed literature reveals some gaps in existing research in terms of how well it covers the hypothesized relations in our framework. These gaps can be translated into requirements for our studies; by covering them, our studies provide a more thorough validation of our framework. Specifically, each empirical study will broadly adhere to a subset of the following requirements:

### 3.7.1 Requirements regarding objective system aspects (*manipulations*)

1. *Algorithm*: Despite the predominant research interest in algorithms, there is an apparent paucity of knowledge on how algorithm accuracy influences user experience. The main manipulation in most of our studies is the algorithm used for providing recommendations.
2. *Recommendation set composition*: Another important manipulation is the composition of the set of recommendations presented to the user, as this aspect remains largely untreated in existing research.
3. *Preference input data*: Algorithm and interface meet at the point of preference elicitation. In content-based recommenders this topic has been researched extensively. In collaborative-filtering recommenders, however, the topic remains largely untreated, especially when it comes to explicit versus implicit preference elicitation. Several of our empirical evaluations therefore manipulate the type of input (explicit or implicit) used for recommendation.

### 3.7.2 Requirements regarding subjective system aspects

4. *Perceived aspects as mediators*: Subjective system aspects such as perceived recommendation quality, accuracy and diversity are measured in our studies, as we expect that these perceptions mediate the effect of the objective system aspects on the user experience concepts.

### 3.7.3 Requirements regarding experience and interaction

5. *User experience evaluation*: System effectiveness (system-EXP), choice satisfaction (outcome-EXP) and usage effort and choice difficulty (process-EXP) will measure the three objects of user experience evaluation where possible. Relations between these EXP variables will be reported.
6. *Providing feedback*: Many recommender systems elicit the users' preferences by analyzing their preference feedback. We therefore extensively analyze the positive and negative antecedents of the users' intention to provide feedback, as well as their actual feedback behavior.

7. *Behavioral data:* To link the attitudinal part of user experience to the users' observable behavior, logging data will be triangulated with subjective concepts.

### 3.7.4 Requirements regarding personal and situational characteristics

8. *PC and SC:* Important and under-researched personal and situational characteristics such as domain knowledge and privacy concerns are included where possible and useful.

## 4 Empirical evaluations: validation of the framework

### 4.1 Background

The framework proposed in this paper was originally developed as an analysis tool for the MyMedia project (Meesters et al. 2008), which is part of the European Commission 7th Framework Programme. The main goal of MyMedia was to improve the state-of-the-art of multi-media recommender systems. A recommender system development framework, the MyMedia Software Framework (Marrow et al. 2009),<sup>4</sup> was created and deployed in real-world applications at four industrial partners. Each partner conducted a field trial with their system, with several aims in mind: technical feasibility, business opportunities, and user experience research. In the current paper we discuss the results of four field trials (FT1-FT4), two conducted using the MyMedia version of ClipClub player, developed by the European Microsoft Innovation Center (EMIC), and two conducted using a web-based TV catch-up service, developed by the British Broadcasting Corporation (BBC). Each trial was designed and evaluated on the basis of the proposed evaluation framework. We amended these trials by conducting two experiments (EX1 and EX2) with a comparatively more controlled but also more artificial quality. The tight control over the users' interaction with the system in these experiments allowed us to consider in more detail the decision-making processes underlying the user experience. The six studies are described in Table 1, and will be discussed in more detail below.

The main goal of the studies is two-fold: To generate new knowledge that fills the gaps in current research, and to validate the proposed evaluation framework. To assist this latter goal, we repeatedly include the recommendation algorithm (an OSA), the perceived recommendation quality (an SSA), and the system's effectiveness (EXP) as core components of our analysis. Furthermore, for each study we discuss the extent to which its results fit the framework.

We realize that the exclusive focus on collaborative filtering recommender systems, all based on the same MyMedia development framework, and all with media content, limits the scope of this validation. Despite this, we believe that the evaluation framework itself has sufficiently generic qualities to apply to recommender systems

<sup>4</sup> The recommender algorithms used in the studies described here are available in the MyMedia Software Framework, which is available for non-commercial research purposes, <http://mymediaproject.codeplex.com>, as well as in the open source package MyMediaLite, <http://isml.de/mymedialite>.

**Table 1** Properties of the studies that were conducted based on the evaluation framework*FT1 Emic pre-trial*

|                |   |
|----------------|---|
| System         | Adjusted Microsoft ClipClub   |
| Content        | Continuously updated database of clips targeted at teenagers  |
| Participants   | 43 EMIC colleagues and partners   |
| Manipulations  | Algorithms: <ul style="list-style-type: none"> <li>• Random recommendations</li> <li>• Vector Space Model (VSM) algorithm based on explicit feedback</li> </ul>   |
| Main questions | Does a system that provides personalized recommendations provide a better user experience than a system that provides random recommendations?<br>What factors influence the users' intention to provide feedback? |

*FT2 EMIC trial*

|                |  |
|----------------|--|
| System         | Adjusted Microsoft Clipclub  |
| Content        | Continuously updated database of clips targeted at teenagers   |
| Participants   | 108 externally recruited "young" participants (targeted mean age of 25)  |
| Manipulations  | Algorithms: <ul style="list-style-type: none"> <li>• General most popular items</li> <li>• Bayesian Personalized Ranking Matrix Factorization (BPR-MF) algorithm based on implicit feedback</li> <li>• VSM algorithm based on explicit feedback</li> </ul> Scenario: <ul style="list-style-type: none"> <li>• Users receive no specific information</li> <li>• Users are told that their ratings are collected and that this data is used to provide better recommendations</li> <li>• Users are told that their behavior is monitored and that this data is used to provide better recommendations</li> </ul> |
| Main questions | What is the difference in subjective recommendation quality between the different algorithms?<br>Does a system that provides personalized recommendations lead to a better user experience than a system that recommends the "generally most popular" items?<br>What is the difference between the implicit recommender and the explicit recommender in terms of user experience?<br>What factors influence the users' intention to provide feedback?  |

*FT3 BBC pre-trial*

|                |   |
|----------------|---|
| System         | BBC MyMedia player  |
| Content        | BBC television programming (up to one week old)   |
| Participants   | 59 externally recruited British participants, reflecting a balanced representation of the UK television audience  |
| Manipulations  | For the rating trial, algorithms: <ul style="list-style-type: none"> <li>• General most popular items</li> <li>• BPR-MF algorithm based on implicit feedback</li> <li>• MF algorithm based on explicit feedback</li> </ul> For the rating trial, time: <ul style="list-style-type: none"> <li>• Day 1 ... Day 9</li> </ul> For the experience trial, time: <ul style="list-style-type: none"> <li>• Week 1</li> <li>• Week 2</li> </ul> |
| Main questions | What is the difference in recommendation list quality between the different algorithms?<br>How does the quality of the recommendation lists generated by the different algorithms evolve over time?<br>How does the user experience of the system evolve over time?   |

**Table 1** continued

| <i>FT 4 BBC trial</i>                 |  |
|---------------------------------------|--|
| System                                | BBC MyMedia player   |
| Content                               | BBC television programming (up to one week old)  |
| Participants                          | 58 externally recruited British participants, reflecting a balanced representation of the UK television audience   |
| Manipulations                         | Algorithms: <ul style="list-style-type: none"> <li>• General most popular items</li> <li>• BPR-MF algorithm based on implicit feedback</li> <li>• MF algorithm based on explicit feedback</li> </ul>   |
| Main questions                        | What is the difference in subjective recommendation quality between the different algorithms?<br>Does a system that provides personalized recommendations lead to a better user experience than a system that recommends the “generally most popular” items?<br>What is the difference between the implicit recommender and the explicit recommender in terms of user experience?  |
| <i>EX1 Choice overload experiment</i> |  |
| System                                | Adjusted BBC MyMedia player  |
| Content                               | Movies (MovieLens 1M dataset)  |
| Participants                          | 174 participants invited from a panel with students or recently graduated students from several Dutch universities   |
| Manipulations                         | Recommendation set quality and size: <ul style="list-style-type: none"> <li>• Top-5 (5 best recommendations)</li> <li>• Top-20 (20 best recommendations)</li> <li>• Lin-20 (5 best recommendations, recommendation ranked 99, 199, ... 1499)</li> </ul>  |
| Main questions                        | How do the objective quality and size of the recommendation set influence the subjective recommendation set quality and diversity?<br>How do the objective quality and size of the recommendation set influence choice difficulty and choice satisfaction?   |
| <i>EX2 Diversification experiment</i> |  |
| System                                | Adjusted BBC MyMedia player  |
| Content                               | Movies (MovieLens 1M dataset)  |
| Participants                          | 137 Amazon Turk workers  |
| Manipulations                         | Algorithms: <ul style="list-style-type: none"> <li>• General most popular items</li> <li>• MF algorithm based on explicit feedback</li> <li>• K-Nearest Neighbor (kNN) algorithm based on explicit feedback</li> </ul> Recommendation set diversification: <ul style="list-style-type: none"> <li>• No diversification</li> <li>• Some diversification</li> <li>• Lots of diversification</li> </ul>   |
| Main questions                        | What is the difference in subjective recommendation quality between the different algorithms?<br>Does a system that provides personalized recommendations lead to a better user experience than a system that recommends the “generally most popular” items?<br>What is the difference between the kNN recommender and the MF recommender in terms of user experience?<br>Do users notice our manipulation of the variety of the recommendations?<br>Do users like (objective or subjective) variety in their recommendation sets? If so, does the effect overshadow the effect of algorithm accuracy? |

in general. The framework is based on a broad range of existing research, and does not assume specific operationalizations of the measured concepts. Furthermore, the manipulations and main questions in the conducted studies are rather generic, and hence the range of validity of our conclusions can be broadened from multi-media recommender systems to recommender systems in general.

## 4.2 Empirical validation techniques

User experience research can be conducted both qualitatively and quantitatively (Preece et al. 2002; Kaplan and Duchon 1988). Qualitative research is more exploratory, but is usually less generalizable, and cannot be statistically validated. Quantitative analysis allows for statistical validation, but one has to have clear hypotheses about theoretical constructs and their relations before conducting the study. In many cases it is advisable to apply these techniques in tandem, but for the purpose of this paper we will restrict our analysis to quantitative results (hypotheses for our studies are conveniently provided by the framework). In order to prevent confirmation bias in our validation of the framework, we extensively use data analysis methods like exploratory factor analysis (EFA) and structural equation modeling (SEM), which allow for a more exploratory analysis of quantitative data. Below we discuss the general procedure of our research; Appendix A provides a more in-depth description. We recommend researchers who want to use our framework to either follow a similar procedure, or to opt for our pragmatic version described elsewhere (Knijnenburg et al. 2011a).

### 4.2.1 Measurement

Experience, SSAs, PCs and SCs can be measured using questionnaires. To assure a more robust measurement of all concepts, we typically use a minimum of seven statements (both positively and negatively phrased) that can be answered on a balanced 5- or 7-point scale (from “completely disagree” to “completely agree”) for each unidimensional concept. Exploratory factor analyses can be conducted to test the robustness of the concepts, and to exclude any questions that do not contribute to the measurement of the intended concepts.

### 4.2.2 Manipulation

Objective system aspects (OSA) can be manipulated, i.e., several versions of the aspect (conditions) can be created, and assigned randomly to each participant. By keeping everything else constant between conditions, one can single out the effect of this manipulated aspect on the user experience (EXP). In our research, one condition always serves as a baseline against which all other conditions are compared.<sup>5</sup> Furthermore, by manipulating several system aspects independently, one can compare the relative impact of these aspects on the user experience (e.g. “Does the effect of the user interface overshadow the impact of different algorithms?”).

<sup>5</sup> Similar to the use of dummy variables in standard linear regression.

#### 4.2.3 Structure

The incorporation of user perceptions (SSA) increases the robustness and explanatory power of the evaluations, and different objects of user experience (system-EXP, process-EXP, outcome-EXP) can be treated in parallel to gain further insight in the workings of user experience. Logged behavioral data can be triangulated with the user experience concepts to link the subjective experience (EXP) to objective user behavior (INT). This creates a causal chain of effects from manipulated OSAs, via subjectively measured SSAs and EXPs, to objectively measured INTs. Structural equation modeling (SEM; [Muthen 1984](#)) can be used to conduct a mediation analysis on these effects. From a statistical perspective, SEM concurrently tests the robustness of the measured constructs and the relationships between them.

#### 4.2.4 Graphical presentation of SEMs

We graphically present our structural equation models as diagrams containing the constructs (boxes) and the relationships between them (arrows). Rectangular boxes are latent constructs based on the questionnaire items. For the sake of clarity, we do not include the questionnaire items themselves in the diagrams. Elliptical boxes are behavioral metrics, extracted from the systems' data logs. One-headed arrows represent regression coefficients (which have a causal direction); double-headed arrows represent correlation coefficients.

The color of the boxes matches the colors in the framework (Fig. 1); each color signifies a specific type of construct (purple: OSA, green: SSA, orange: EXP, blue: INT, red: PC, light blue: SC). The numbers on the arrows represent the regression coefficients (i.e. the strength of the structural relations, also represented by the thickness of the arrows), their standard deviation (between parentheses) and the statistical significance. Non-significant relations are not included in the graphs.

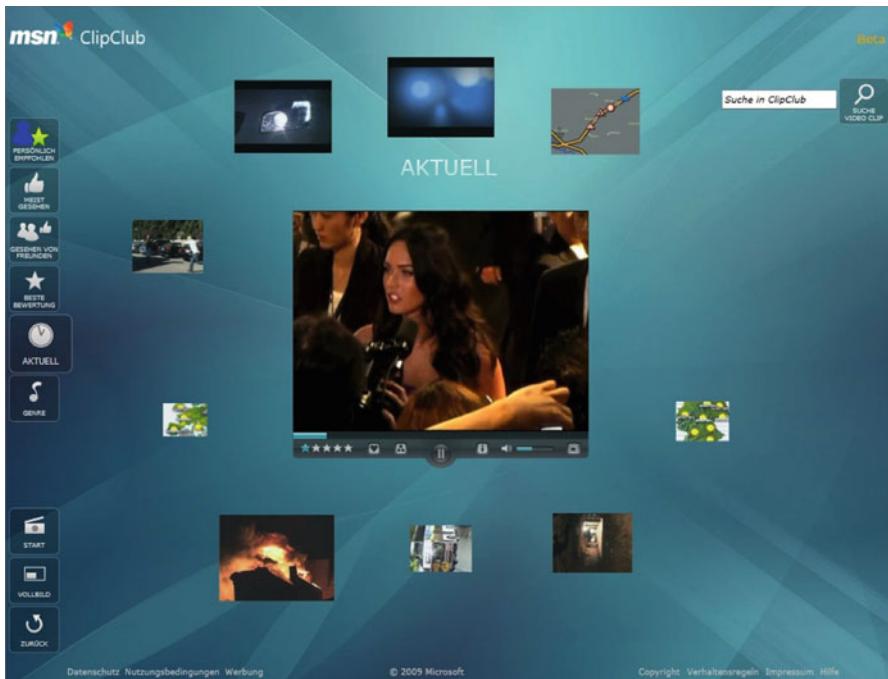
### 4.3 FT1 EMIC pre-trial

The EMIC pre-trial was conducted to confirm two basic premises for the success of recommender systems in general: the premise that the user experience of a recommender is influenced by the recommendations, and the premise that users will provide adequate preference feedback to train the recommender system.

#### 4.3.1 Setup

The trial was conducted with 43 EMIC colleagues and partners who participated in the trial on a voluntary basis (28 male, average age of 31,  $SD = 9.45$ ). A detailed treatment of this study can be found in [Knijnenburg et al. \(2010b\)](#). We therefore only briefly discuss the results here.

Participants all used a slightly modified version of the MSN Clipclub system (see Fig. 2); the special section with recommendations was highlighted, the social networking features were disabled (as to not interfere with the study), an explanation of the



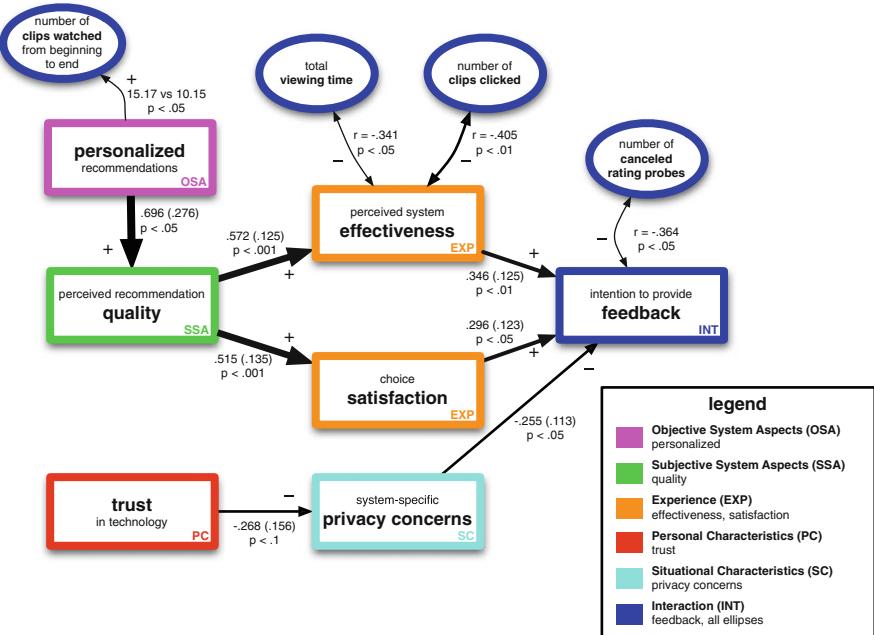
**Fig. 2** The modified ClipClub prototype

rating facilities was included, and participants were probed to provide at least one rating every five minutes (although this rating request could be denied). Participants were randomly assigned to one of two conditions: 25 participants received random clips as recommendations, and the remaining 18 participants received recommendations provided by a content-based Vector Space Modeling engine (i.e. we manipulated the OSA “personalized vs. random recommendations”). Participants were told that providing ratings would change the recommendations.<sup>6</sup>

After half an hour of interaction with the system, several questionnaires<sup>7</sup> were taken to measure the participants’ perceived recommendation quality (SSA), choice satisfaction (outcome-EXP), perceived system effectiveness (system-EXP), intention to provide feedback (INT), general trust in technology (PC), and system-specific privacy concerns (SC). The questions were factor-analyzed to produce metrics for these concepts, and factor scores were included together with the manipulation in a regression path model (Fig. 3). Additionally, factor scores were correlated with behavioral metrics obtained from click-stream logs.

<sup>6</sup> Which is true for both conditions, but in the random condition these new recommendations would just be another batch of random items.

<sup>7</sup> A complete overview of the questionnaires used in the studies can be found in Appendix B.



**Fig. 3** The path model constructed for FT1 - EMIC pre-trial. Please refer to Sect. 4.2.4 “Graphical presentation of SEMs” for interpretation of this figure. Note that the rectangle “personalized recommendations” represents the difference between personalized and random recommendations, random being the baseline model

#### 4.3.2 Results

The results show that personalized recommendations (as compared to random recommendations) have a higher perceived quality (OSA → SSA), which leads to a higher choice satisfaction (SSA → outcome-EXP) and system effectiveness (SSA → system-EXP). Behavioral data corroborates these hypotheses. Users of the system with personalized recommendations watch a larger number of clips from beginning to end (OSA → INT). Moreover, users who click fewer clips and have a lower total viewing time rate the system as more effective (EXP ↔ INT), which indicates that a higher system effectiveness is related to reduced browsing activity (see discussion below).

The intention to provide feedback increases with choice satisfaction and system effectiveness (EXP → INT) but decreases when users have a higher system-specific privacy concern (SC → INT), which in turn increases when they have a lower trust in technology (PC → SC).<sup>8</sup> In terms of behavior, the number of canceled rating probes (popping up after five minutes without rating) is significantly lower in the personalized condition than in the random condition (OSA → INT), and is also negatively correlated with intention to provide feedback (INT ↔ INT).

<sup>8</sup> An effect of a personal characteristic on a situational characteristic is not explicitly predicted by our framework, but in this case makes perfect sense.

### 4.3.3 Discussion of results

The results show that a system with a recommender algorithm that provides personalized recommendations has a better user experience (in terms of both choice satisfaction and system effectiveness) than a system that provides random recommendations. This is not necessarily a surprising result; such a comparison between random and personalized recommendations is a bit unbalanced ([Van Velsen et al. 2008](#)).

More interestingly, however, the path model indicates that this effect is indeed mediated by perceived recommendation quality (requirement 4 in Sect. [3.7](#)). Furthermore, there is no residual correlation between choice satisfaction and system effectiveness, and a mediated variant provides a weaker model than the one described. In other words, in this study there was no structural relation between these experience variables (requirement 5).

Users seem to base their intention to provide feedback on a trade-off between having a better user experience and maintaining their privacy (requirement 6 and 8). However, the intention to provide feedback was not correlated with the total number of ratings, indicating that the relation between intention and behavior can be very weak (a well-known psychological phenomenon called the ‘intention-behavior gap’; [Sheeran 2002](#)).

User behavior is correlated with the experience variables (requirement 7), but at times also directly with the manipulation of our study. Against our intuition, users who rate the system as more effective have a lower total viewing time and a lower number of watched clips. However, the results also show that at the same time, the number of clips watched from beginning to end is higher in the personalized condition than in the non-personalized condition. The lower total viewing time and number of clicked clips thus reflects a reduction in browsing, not consumption. This makes sense, because recommendations are supposed to be an alternative to browsing in this system. The outcomes clearly demonstrate the value of triangulating the behavioral measures with subjective measures. Showing how behavioral measures are related to experience variables allows researchers to assign meaning to these measurements, which at times may even counter the researchers’ intuition. Moreover, it grounds our subjective theory in observable behavior.

## 4.4 FT2 EMIC trial

The EMIC extended trial was conducted to reproduce and extend the effects of the pre-trial. Taking a step beyond the unbalanced comparison in the pre-trial between personalized and random recommendations, the extended trial looked at differences between the non-personalized “generally most popular” items (GMP condition), the VSM algorithm using explicit feedback as input (the same VSM condition as used in the pre-trial) and the Bayesian Personalized Ranking Matrix Factorization algorithm (BPR-MF; [Rendle et al. 2009](#)), a state-of-the-art algorithm using all clicks to predict recommendations (‘implicit feedback’; MF-I condition). Furthermore, we specifically controlled what the users were told about the systems’ use of their feedback: nothing (none condition); that their rating behavior was being used to provide better

recommendations (rating behavior condition); or that all their behavior was being used to provide better recommendations (all behavior condition). We hypothesized that this manipulation would influence users' privacy concerns.

#### 4.4.1 Setup

An external company recruited German participants from a young demographic (targeted mean age of 25). Participants were instructed to use the system as many times as they liked over the 20-day duration of the study, and they were asked to fill out a 47-item user experience questionnaire after each session (which would comprise at least 20 minutes of interaction). The external company paid the participants for their cooperation. Participants were randomly assigned to a scenario (i.e. the OSA "scenario" [no story/rating behavior/all behavior] is a between subjects manipulation), and after each questionnaire they would switch algorithms (i.e. the OSA "algorithm" [GMP/VSM/MF-I] is a within subjects manipulation). Participants were informed of this switch, but were not told which algorithm they would be using. Users used the same system as in FT1 (see Fig. 2), but in contrast to the first field trial, there were no explicit rating requests in this trial, and there was also no specific section with recommendations; instead, all categories that the user could navigate to (e.g. sport, gossip, cars, news) showed a personalized subset of the items in that category. The behavior of each participant was logged, allowing for an extensive analysis of the click-stream of each user.

The trial yielded 430 questionnaires from 108 participants. After excluding all questionnaires with fewer than 12 clicks (indicating insignificant usage), 258 questionnaires (60%) remained from 95 remaining participants (88%). These participants had an average age of 27.8 ( $SD = 4.70$ ). 49 of them were male.

The questions in the questionnaires were first submitted to an exploratory factor analysis (EFA) to determine whether their covariances naturally reproduced the predicted constructs. This resulted in 7 factors:

- Perceived recommendation quality (6 items, e.g. "I liked the items shown by the system", factor  $R^2 = .009$ )<sup>9</sup>
- Effort of using the system (3 items, e.g. "The system is convenient", factor  $R^2 = .184$ )
- Perceived system effectiveness and fun (10 items, e.g. "I have fun when I'm using the system", factor  $R^2 = .694$ )<sup>10</sup>
- Choice satisfaction (5 items, e.g. "I like the items I've chosen", factor  $R^2 = .715$ )
- General trust in technology (4 items, e.g. "Technology never works", no incoming arrows)

<sup>9</sup>  $R^2$  values are taken from the final SEM and not from the EFA. The  $R^2$  for perceived recommendation quality was low because it was predicted by the algorithm condition only. We typically report the best fitting item as an example for the scale, full questionnaires can be found in Appendix B.

<sup>10</sup> Fun was intended to be a separate construct, but the exploratory factor analysis could not distinguish this construct from perceived system effectiveness. In other words, in answering our questionnaires participants did not seem to conceptually distinguish these two constructs.

- System-specific privacy concerns (3 items, e.g. “The system invades my privacy”, factor  $R^2 = .333$ )
- Intention to provide feedback (4 items, e.g. “I like to give feedback on the items I’m watching”, factor  $R^2 = .264$ ).

12 items were deleted due to low communalities and/or unwanted cross-loadings. The items were then analyzed using a confirmatory structural equation modeling (SEM) approach with repeated ordinal dependent variables and a weighted least squares estimator, in which the subjective constructs were structurally related to each other, to the conditions (algorithm and scenario), and to several behavioral measures extracted from the usage logs. The final model had a reasonable model fit ( $\chi^2(41) = 85.442$ ,  $p < .001$ ,  $CFI = .977$ ,  $TLI = .984$ ,  $rMSEA = .065$ ).<sup>11</sup> Figure 4 displays the effects found with this model.

#### 4.4.2 Results

The model shows that the recommendations from the Matrix Factorization algorithm (MF-I) have a higher perceived quality than the non-personalized “generally most popular” (GMP) items (OSA → SSA). Higher perceived recommendation quality in turn leads to lower effort (SSA → process-EXP), a higher perceived effectiveness and fun (SSA → system-EXP), and a higher choice satisfaction (SSA → outcome-EXP). The effect of the algorithm on choice satisfaction is however only partially mediated; there is also a direct positive effect of the MF-I algorithm on choice satisfaction (OSA → EXP). The vector space modeling algorithm (VSM) does not even affect perceived quality at all; it only has a direct effect on choice satisfaction (increasing it marginally;  $p < .10$ ). Lower effort leads to more perceived effectiveness and fun (process-EXP → system-EXP; this is in line with Pommeranz et al. 2012), which in turn leads to more satisfactory choices (system-EXP → outcome-EXP).

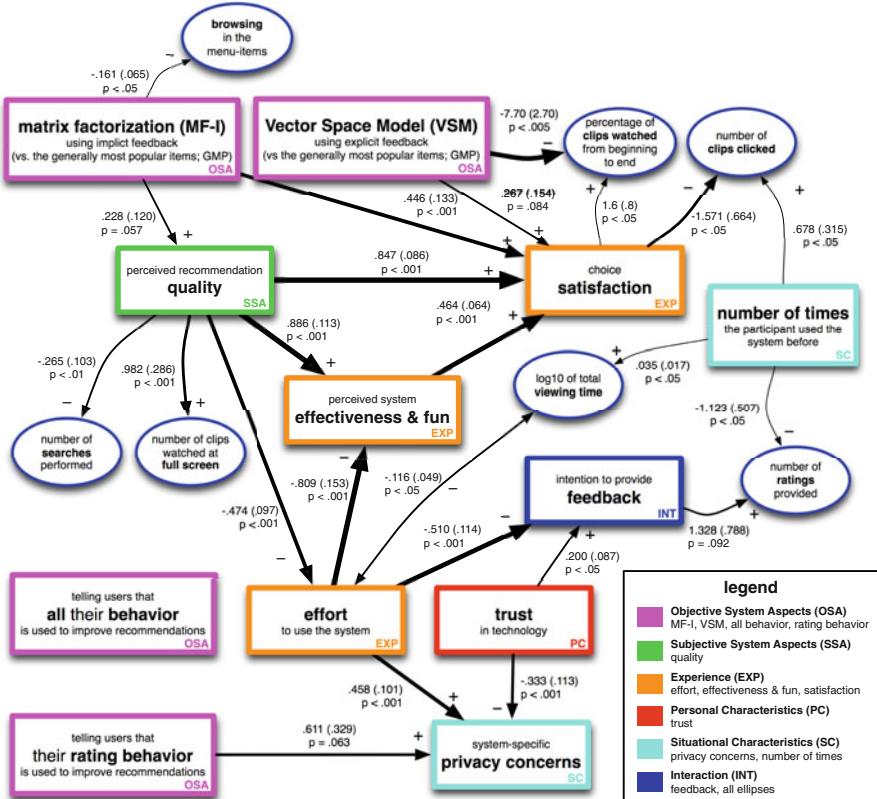
The effort required to use the system influences the intention to provide feedback: the less effort users have to invest, the more they are willing to provide feedback (process-EXP → INT). Privacy concerns also increase when the system takes more effort to use (process-EXP → SC).<sup>12</sup> Figure 5 displays the tendency of marginal effects of the different algorithms on recommendation quality, system effectiveness & fun, and choice satisfaction. The graphs show the standardized difference between the algorithms (MF-I and VSM) and the baseline condition (GMP).

Figure 4 shows a direct effect of trust in technology on users’ intention to provide feedback (PC → INT). Trust in technology also reduces privacy concerns (PC → SC). Telling users that their rating behavior is used to improve recommendations increases their privacy concerns (OSA → SC).<sup>13</sup> Notably, telling users that all their behavior is being used to improve recommendations does not have this effect.

<sup>11</sup> Agreed-upon model fit requirements are described in more detail in Appendix A.

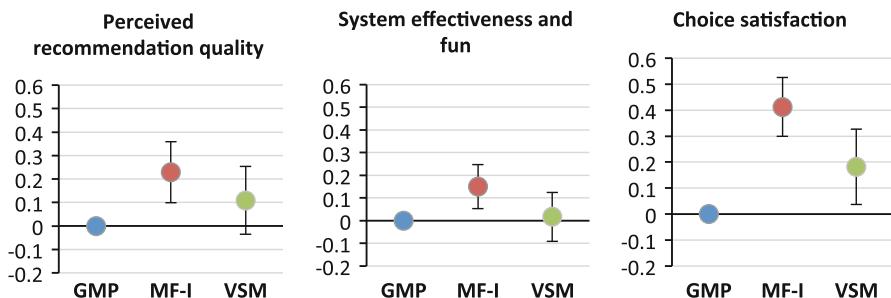
<sup>12</sup> Our framework does not predict an effect of experience on situational characteristics. However, we understand that privacy concerns may be higher when users have to put more effort into using the system, because this usually means that they also have to provide more information to the system.

<sup>13</sup> Our framework does not predict an effect from an objective system aspect on a situational characteristic to exist. However, this particular OSA was specifically designed to change this particular SC.



**Fig. 4** The path model constructed for FT2 - EMIC trial. Please refer to Sect. 4.2.4 “Graphical presentation of SEMs” for interpretation of this figure. Note that the “algorithm” manipulation is represented by the two rectangles “Matrix Factorization (MF-I)” and “Vector Space Model (VSM)”, which are tested against the non-personalized baseline “generally most popular (GMP)”. Likewise, the “scenario” manipulation is represented by the two rectangles “all behavior” and “rating behavior”, which are tested against the baseline “no story”

In terms of behavioral measures, we find that, like in the pre-trial (FT1), a high choice satisfaction decreases the number of clips clicked ( $\text{EXP} \rightarrow \text{INT}$ ). Users who perceive the recommendations to be of a higher quality also perform fewer searches ( $\text{SSA} \rightarrow \text{INT}$ ), and users browse less between different categories when the MF-I algorithm is used ( $\text{OSA} \rightarrow \text{INT}$ ). Users who are more satisfied with their choices watch more clips from beginning to end ( $\text{EXP} \rightarrow \text{INT}$ ). Interestingly, fewer users do so when the VSM algorithm is used ( $\text{OSA} \rightarrow \text{INT}$ ). Users who perceive the recommendations to be of a higher quality also watch more clips at full screen ( $\text{SSA} \rightarrow \text{INT}$ ). Furthermore, intention to provide feedback increases the number of ratings provided by the user, something we did not find in FT1. Finally, in later sessions the number of clips clicked and the viewing time increase ( $\text{SC} \rightarrow \text{INT}$ ). Participants also provide fewer ratings in later sessions ( $\text{SC} \rightarrow \text{INT}$ ; this is in line with [Harper et al. 2005](#)).



**Fig. 5** Tendency of marginal (direct and indirect) effects of the “algorithm” condition on perceived recommendation quality, system effectiveness and fun, and choice satisfaction. *Error bars* indicate  $\pm 1$  standard error compared to the value of GMP, which is fixed to zero. Scales of the vertical axes are in sample standard deviations. The Matrix Factorization algorithm (MF) provides higher perceived recommendation quality and a higher choice satisfaction than the non-personalized “most popular” algorithm (GMP); the Vector Space Modeling algorithm (VSM) does not provide a significantly better experience

#### 4.4.3 Discussion of the results

In general terms, this study confirms the structural relations found in FT1, but there are a number of important differences and additional insights. Specifically, the study allows us to compare two different algorithms with a non-personalized baseline (requirement 1, see Sect. 3.7). The results indicate that the BPR-MF recommendations of the MF-I condition are most successful; these have a marginally higher perceived quality and lead to a higher choice satisfaction (see Fig. 5). The VSM recommendations only lead to a marginally higher choice satisfaction. A possible reason for these differences is that the BPR-MF algorithm used implicit feedback, which can be gathered more quickly than the explicit feedback used by the VSM algorithm (requirement 3).

The inclusion of the construct “effort of using the system” changes part of our path model compared to FT1: Whereas in FT1 choice satisfaction and perceived system effectiveness increased users’ intention to provide feedback, we now observe that it is actually the effort of using the system that causes this effect (requirement 6). This difference between FT1 and FT2 indicates the importance of measuring all aspects of the user experience (requirement 5) as this allows us to better understand the nature of the difference.

The trial confirms that privacy and trust are important factors influencing the intention to provide feedback (requirement 6), although the effects are now structurally different. Like in FT1, trust in technology reduces privacy concerns, but it now also directly increases the intention to provide feedback (instead of a mediated effect via privacy concerns). Due to the increased power of the study, FT2 enables us to measure the link between feedback intention and behavior.

We also observe that telling users that their rating behavior is used (compared to telling them nothing, or that all their behavior is used) increases their privacy concerns. This is surprising, because using all behavior (including rating behavior) is by definition more intrusive than using only rating behavior. The wording in the two conditions is almost the same: “In order to increase recommendation quality, we are going to use [your ratings data]/[all your activities]”. One possible reason for the heightened

privacy concerns in the “rating behavior” scenario is that users in this scenario have active control over the amount of information they provide to the system. They are thus encouraged (or forced) to make an explicit trade-off between the potential usefulness of providing information and the effect this may have on their level of privacy. In the “all behavior” condition, there is nothing the users can do to prevent the system from analyzing their behavior, and it is also difficult to anticipate the privacy effects of exhibited and forborne behavior. Users may therefore be less sensitized with regard to privacy concerns, or even forget that the system is continuously analyzing their behavior.

By triangulating our behavioral data with the subjective constructs (requirement 8), we are also able to revalidate the finding from FT1 that a good experience means: “less browsing, but more time enjoying the content”. Finally, the extended period of use allows us to look at the effect of familiarity with the system. Specifically, users are exploring the system to a further extent in later sessions, which may indicate less reliance on the recommendations.

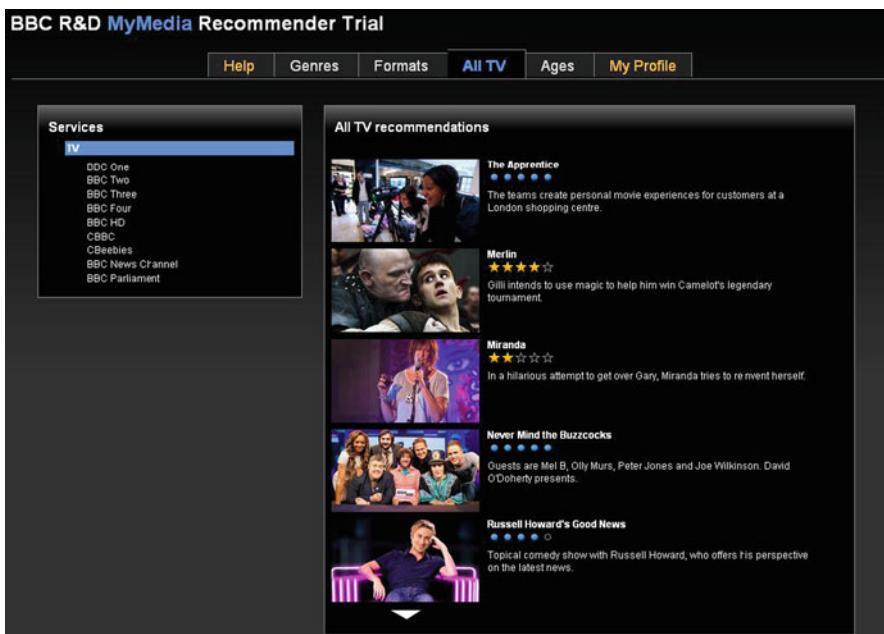
#### 4.5 FT3 BBC pre-trial

The BBC pre-trial used a special-purpose recommender system to present users with recent programming of the BBC (see Fig. 6) and strove to compare three different algorithms side by side. As FT2 revealed that usage over time changes (i.e., we observed less reliance on the recommendations over time), the pre-trial was conducted to investigate how recommendation quality would evolve over time.

##### 4.5.1 Setup

An external market research company recruited 59 British participants to participate in both FT3 and FT4. The participants were selected to reflect a balanced mix of ages (one third between 18 and 35, one third between 36 and 50, and one third between 51 and 66). Within each age group, the selection included equal numbers of males and females. The selected sample also consisted of equal numbers of participants from five regions across the UK. Finally, half of the participants were occasional users of the existing TV/radio catch-up service, the other half were frequent users. Participants were paid a small fee for each daily task, and a bonus for each completed user experience questionnaire. Over a period of two weeks (not counting weekends) participants performed two daily tasks. Specifically, the participants were asked every day to rate a list of recommendations presented by three different algorithms (rating task), and to use the system freely (free use task). The participants had an average age of 41.5 ( $SD = 12.6$ ). 27 of them were male.

The algorithms used in the rating task were the non-personalized ‘generally most popular’ items (the GMP condition), and two personalized Matrix Factorization algorithms. One is described in Rendle and Schmidt-Thieme (2008) and relies on user ratings to provide recommendations (‘explicit feedback’; the MF-E condition). The other, BPR-MF (Rendle et al. 2009), relies on all clicks in the interface (‘implicit feedback’; the MF-I condition). Every day, each participant rated three lists (one from



**Fig. 6** The BBC MyMedia recommender prototype

each of the algorithms) of five recommended programs. The order in which the lists were presented changed every day.

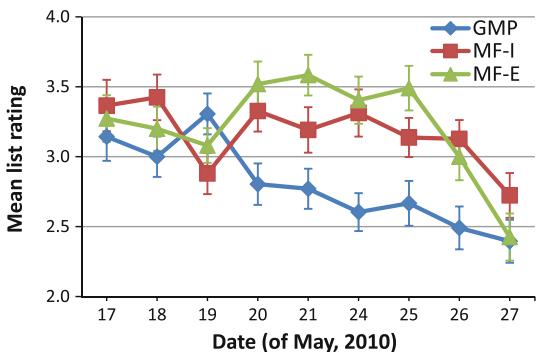
The system in the ‘free use’ task employed the MF-E algorithm for all participants throughout the entire two-week period. After both weeks, participants filled out a questionnaire asking about their user experience with the ‘free use’ system.

#### 4.5.2 Results of the rating task

In the rating task users were asked to rate five recommended programs separately, as well as the recommendation list as a whole. As can be seen in Fig. 7 below, the average ratings decreased over time (contrast  $F(1, 15) = 12.7, p < .005, r = .68$ ). Either participants gradually became more critical about the recommendations (a psychological phenomenon called habituation), or they were actually confronted with recommendations of a lower quality (e.g. because items previously recommended were not included in the recommendations to prevent repetitiveness).

After an initial training period, at days four to seven (May 20–25), we observe significantly higher ratings of the MF-E recommendation lists ( $M_{MF-E} = 3.52$  vs.  $M_{GMP} = 2.80$ , contrast  $F(1, 29) = 45.4, p < .001, r = .78$ ) as well as the MF-I recommendation lists ( $M_{MF-I} = 3.33$  vs.  $M_{GMP} = 2.80$ , contrast  $F(1, 29) = 19.1, p < .001, r = .63$ ). In the second week (May 24–27), the ratings for the lists provided by MF-E and MF-I started to drop significantly (interaction of MF-E vs. GMP with linear decreasing time:  $F(1, 32) = 6.90, p < .05, r = .42$ ; interaction of MF-I vs. GMP with linear decreasing time:  $F(1, 32) = 10.9, p < .005, r = .50$ ), going down to the

**Fig. 7** The means of the list ratings, over time, of the recommendation lists for the generally most popular items (GMP) and the Matrix Factorization algorithm recommendations based on explicit (MF-E) and implicit (MF-I) feedback. The error bars present  $\pm 1$  Standard Error



level of GMP. In other words, the quality of the MF-I and MF-E recommendations improves over time, but later falls back to the level of the non-personalized GMP recommendations.

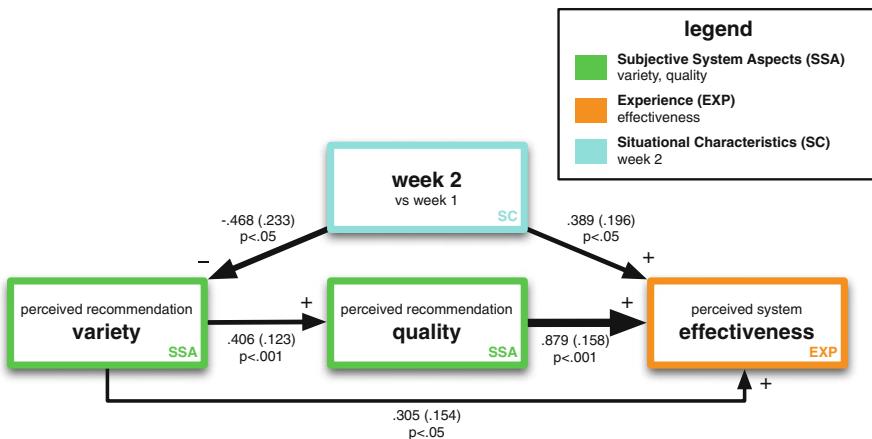
#### 4.5.3 Results of the free use task

To analyze the free use task, we compared the outcomes of the user experience questionnaires collected at the two different time points (i.e. the SC “time” [end of week 1/end of week 2], can be seen as a within subjects manipulation). All 59 users partook in this part of the study, but 8 users only completed the first week’s questionnaire, resulting in 110 data points. The results of the experience questionnaire (17 questions) were first submitted to an exploratory factor analysis (EFA) to see if their covariances naturally reproduced the predicted constructs. This resulted in three factors:

- Perceived recommendation variety (2 questions, “The recommendations contained a lot of variety” and “All the recommended programmes were similar to each other”, factor  $R^2 = .052$ )
- Perceived recommendation quality (7 questions, e.g. “The recommended items were relevant” and “I liked the recommendations provided by the system”, factor  $R^2 = .148$ )
- Perceived system effectiveness (6 questions, e.g. “I would recommend the MyMedia recommender to others” and “The MyMedia recommender is useful”, factor  $R^2 = .548$ ).

Two questions were deleted due to low communalities. The items were then analyzed using a confirmatory SEM approach with repeated ordinal dependent variables and an unweighted least squares estimator, in which the subjective constructs were structurally related to each other and to the dichotomous independent variable ‘time’ (week 1 vs. week 2). The final model had a good model fit ( $\chi^2(22) = 27.258, p = .20, CFI = .982, TLI = .984, rMSEA = .047$ ). Figure 8 displays the effects found with this model.

The model shows that a higher perceived variety of the recommendations causes users to perceive the recommendations as having a higher quality, and that higher



**Fig. 8** The path model constructed for FT3 - BBC pre-trial. Please refer to Sect. 4.2.4 “Graphical presentation of SEMs” for interpretation of this figure

quality recommendations in turn cause an increased perceived system effectiveness ( $\text{SSA} \rightarrow \text{SSA} \rightarrow \text{EXP}$ ). There is also a direct effect of perceived variety on perceived system effectiveness ( $\text{SSA} \rightarrow \text{EXP}$ ); the perceived quality does not fully mediate the effect of variety on effectiveness.

Additionally, in this study the variety of the recommendations turned out to be significantly lower in the second week ( $\text{SC} \rightarrow \text{SSA}$ ), which in turn led to a lower perceived quality of the recommendations in the second week. This is in line with the drop in recommendation list ratings as found in the rating task (see Fig. 7). The lower perceived recommendation quality, in turn, decreased the perceived effectiveness of the system. However, time also had a positive direct effect on perceived system effectiveness ( $\text{SC} \rightarrow \text{EXP}$ ), which cancelled out the negative indirect effect (the mean system effectiveness did not differ significantly between week 1 and week 2).

#### 4.5.4 Discussion of the results

The drop in perceived recommendation quality in the second week was consistent between the rating and free use task. The SEM model explains that this happened because the set of recommended programs was less varied in the second week. Arguably, perceived variety drops because TV programs repeat after the first week, at which point they resurface among the recommendations. Interestingly, due to a direct positive effect of time on perceived system effectiveness, these effects in the end did not reduce the user experience. An ad hoc explanation of this effect could be that although users may have felt that recommending new episodes of previously recommended shows is not a sign of high quality recommendations, they may at the same time have appreciated the reminder to watch the new episode. The study thus clearly shows the merit of measuring subjective concepts in addition to user ratings in understanding the underlying causes of rating differences.

## 4.6 FT4 BBC trial

The BBC trial was primarily conducted to test the effect of explicit versus implicit preference elicitation methods on the user experience of the system.

### 4.6.1 Setup

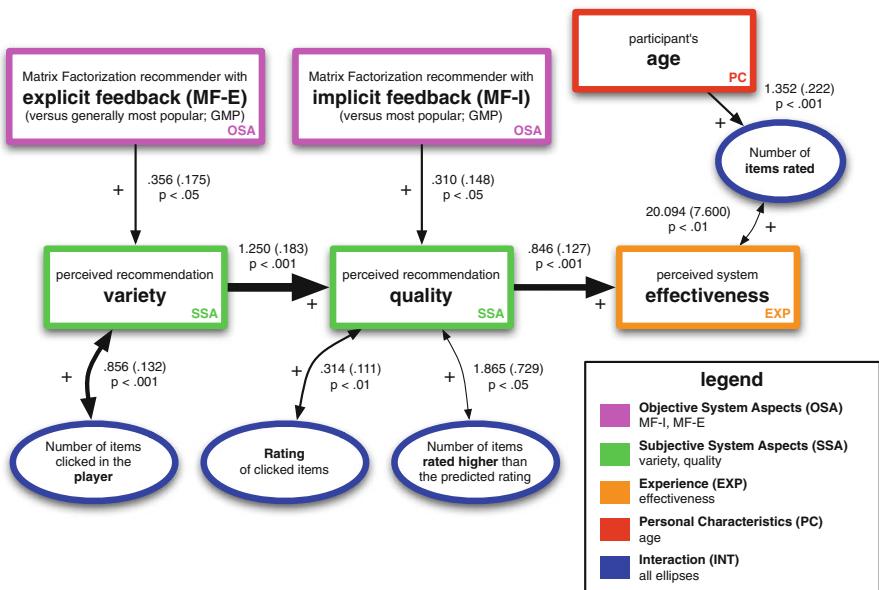
The trial used the same system as the ‘free use’ system in the pre-trial, but now the algorithm that provided the recommendations in this system was changed every three days between GMP, MF-I and MF-E until each algorithm was used once by each user (see Sect. 4.5.1 for more details on these conditions). In each condition users were given the same background on the system, specifically, no information was given on the way recommendations were computed or on what type of information they were based. Participants were randomly assigned to one of three groups, for which the order of the algorithms was manipulated in a Latin square design, so that each algorithm was being used by one third of the users at any time. 58 users (the same as those in the pre-trial) completed this study, resulting in 174 data points. The user experience questionnaire was similar to the pre-trial, but two questions were added to gain a more robust measurement of recommendation variety (now measured by 4 items). The exploratory factor analysis resulted in the same three factors (see Sect. 4.5.3). The behavior of each participant was logged, allowing for an extensive analysis of the click-stream of each user.

The questionnaire items were then analyzed using a confirmatory structural equation modeling (SEM) approach with repeated ordinal dependent variables and a weighted least squares estimator, in which the subjective constructs were structurally related to each other, to the conditions (algorithm/preference input method), and to six behavioral measures extracted from the usage logs. The final model (Fig. 9) had a reasonably good fit ( $\chi^2(29) = 49.672$ ,  $p = .0098$ ,  $CFI = .976$ ,  $TLI = .982$ ,  $rMSEA = .065$ ).

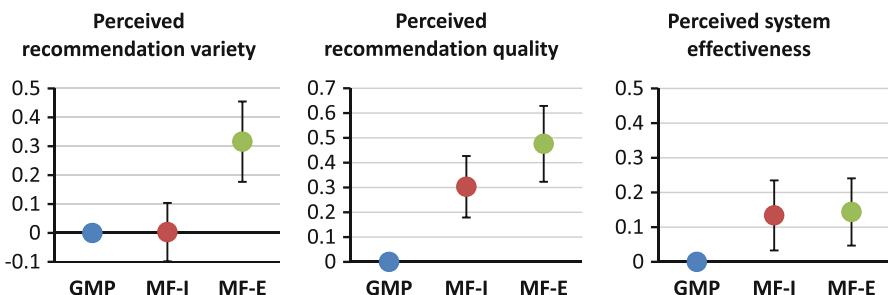
### 4.6.2 Results

As in FT3 (see Fig. 8), we observe that a higher perceived recommendation variety leads to a higher perceived recommendation quality, which in turn leads to a higher perceived system effectiveness (SSA → SSA → system-EXP). Interestingly, the two variants of the matrix factorization algorithm affect different SSAs: compared to GMP, MF-E recommendations (which are based on explicit feedback) have a significantly higher perceived variety, while MF-I recommendations (which are based on implicit feedback) have a significantly higher perceived quality (OSA → SSA). Despite their different trajectories, the net effect of these algorithms is that their increased perceived quality positively affects the user experience (in terms of perceived system effectiveness) compared to the GMP condition (see also Fig. 10).

In terms of behavioral measures, perceived variety is correlated with the number of items clicked in the player (as opposed to the catalog; INT → SSA). Recommendation quality is correlated with the rating users give to clicked items, and the number



**Fig. 9** The path model constructed for FT4 - BBC trial. Please refer to Sect. 4.2.4 “Graphical presentation of SEMs” for interpretation of this figure. Note that the “preference elicitation method” manipulation is represented by the two rectangles “explicit feedback (MF-E)” and “implicit feedback (MF-I)”, which are tested against the non-personalized baseline “generally most popular” (GMP)”



**Fig. 10** Tendency of marginal (direct and indirect) effects of the “preference elicitation method” condition on perceived recommendation variety, perceived recommendation quality, and system effectiveness. Error bars indicate  $\pm 1$  Standard Error. The value of GMP is fixed to zero; scales are in sample standard deviations

of items that are rated higher than the predicted rating (SSA  $\rightarrow$  INT).<sup>14</sup> Finally, the perceived system effectiveness is correlated with the total number of items that participants rate (EXP  $\leftrightarrow$  INT). We furthermore note that older participants rate significantly more items (PC  $\rightarrow$  INT).

<sup>14</sup> SSA  $\rightarrow$  INT is not predicted, but these behaviors are a direct expression of the SSA.

#### 4.6.3 Discussion of the results

This study confirms the basic structural relations between manipulation, subjective system aspects, and experience as also found in FT3 ( $\text{OSA} \rightarrow \text{SSA} \rightarrow \text{EXP}$ ; requirement 4, see Sect. 3.7). In this case, the OSA is the preference elicitation method (requirement 3). Two variants of the matrix factorization algorithm using different types of input (MF-E and MF-I) are tested against a non-personalized baseline (GMP). The fact that both MF-E and MF-I resulted in a better user experience than GMP is not very surprising, but interestingly the model shows that the cause for this effect is different for each algorithm. MF-E (which makes recommendations based on explicit feedback) seems to result in a higher variety in recommendations than GMP, which in turn leads to a higher recommendation quality and a higher perceived effectiveness ( $\text{OSA} \rightarrow \text{SSA} \rightarrow \text{EXP}$ ); MF-I (which makes recommendations based on implicit feedback) on the other hand seems to have a direct influence on recommendation quality, which also leads to a higher perceived effectiveness ( $\text{OSA} \rightarrow \text{SSA} \rightarrow \text{EXP}$ ).

The lack of an increased diversity for MF-I recommendations could be caused by a “pigeonholing effect”: when the MF-I algorithm uses clicks to predict implicit preferences, and gives recommendations based on these preferences, most subsequent clicks will be in line with these predicted preferences, and the algorithm will increasingly home in on a very specific set of recommendations. In MF-E, negative explicit feedback (low ratings) can prevent such pigeonholing. Like Jones et al. (2009), our results show that explicit control over the recommendations leads to a higher recommendation quality through an increase of perceived variety, but that *beyond* the effect through variety, there is no increase in recommendation quality (e.g. by means of a higher accuracy). The marginal effect on the perceived quality of the recommendations for MF-E and MF-I is about equal, and higher than the quality of the non-personalized recommendations (GMP). The same holds for perceived system effectiveness (see Fig. 10).

An interesting way to exploit the difference between MF-I and MF-E would be to combine this result with the result of FT3, which found that diversity decreases over time (see Fig. 8). In this respect, a recommender system could start out with the MF-I algorithm, which can provide high-quality recommendations from the start (as it does not need any ratings). As the diversity of the recommendations starts decreasing, the algorithm can put more weight on the users’ ratings in determining their preferences, or even switch to the MF-E algorithm altogether. This would give the system a boost in recommendation variety, which should provide a better user experience in the long run.

Several behavioral correlates corroborate our subjective measures (requirement 7). The perceived variety of the recommendations is related to the number of items clicked in the player. These items are related to the item that is currently playing, and are therefore arguably more specialized (and thus over time more varied) than other recommendations. The results further show that users reward good recommendations with higher ratings, and system effectiveness is correlated with the number of ratings the user provides (see also requirement 6). The causal effect here is unclear: participants that rate more items may end up getting a better experience because the algorithms gain more knowledge about the user ( $\text{INT} \rightarrow \text{EXP}$ ). On the other hand, as

found in FT1 (see Fig. 3), users may intend to rate more items when they notice that this improves their experience (EXP → INT).

#### 4.7 EX1 choice overload experiment

The four studies described above are all field trials, testing real-life user-behavior in real-world systems. These studies have a high ecological validity, but it is hard to analyze decision processes in detail, because users are unrestricted in their needs, goals, and actions. To get more in-depth knowledge about users' decision processes, we conducted two controlled lab experiments. The tasks in these experiments are fixed, as is the interaction with the system: Users first rate a set of test-items and then make a single decision, each considering the same choice goal. The first of these experiments looked at the effect of the size and quality of the recommendation set on the user experience to study a phenomenon called choice overload. The experiment is described in detail in (Bollen et al. 2010), so here we will only briefly discuss the results and the merit of using the framework in this experiment.

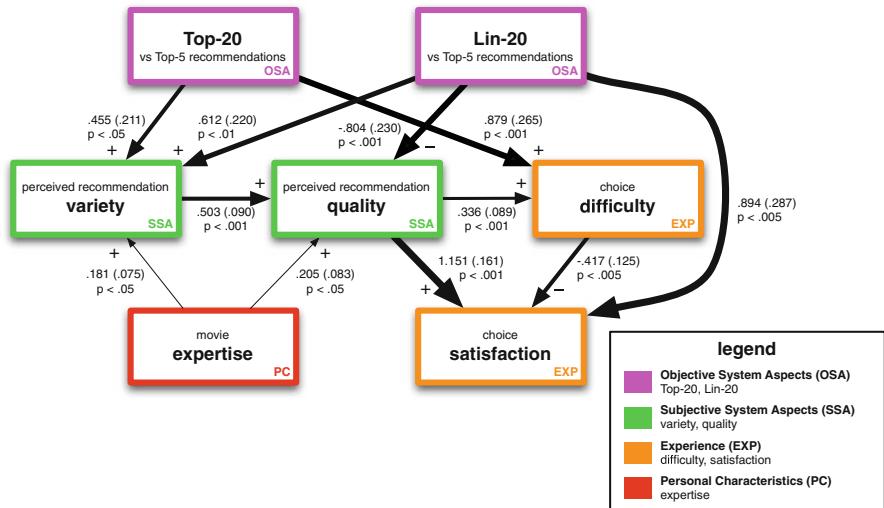
##### 4.7.1 Setup

The experiment was conducted with 174 Dutch participants (invited from a panel with students or recently graduated students from several Dutch universities) with an average age of 26.8( $SD = 8.6$ ). 89 of them were male. Participants were paid €3 upon successful completion of the experiment. They were asked to use the system (loaded with the 1M MovieLens dataset,<sup>15</sup> and using the Matrix Factorization algorithm implementation from the MyMedia Software Framework) to find a good movie to watch. To train the recommender system, they first rated at least 10 movies they knew, and were subsequently presented with a list of recommendations from which to make a choice. Users were randomly assigned to receive either one of three recommendation sets: Top-5 (the best five recommendations, according to our MF algorithm), Top-20 (the best twenty recommendations), and Lin-20 (the best five recommendations, supplemented with the recommendations with rank 99, 199, 299, ..., 1499). After making a choice, users completed a set of questionnaires to measure perceived recommendation set variety, recommendation set attractiveness, choice difficulty, satisfaction with the chosen item, and movie expertise. A Structural Equation Model was fitted on these questionnaire constructs and our two dichotomous manipulation variables ("Top-20 vs. Top-5" and "Lin-20 vs. Top-5"); Fig. 11 shows the resulting path model; for details on the data analysis, refer to (Bollen et al. 2010).

##### 4.7.2 Results

The model shows that satisfaction with the chosen item (outcome-EXP) is the result of two opposing forces: a positive effect of the quality of the recommendations (SSA

<sup>15</sup> The MovieLens datasets are freely available at <http://grouplens.org>. The 1M MovieLens dataset contains one million ratings for 3952 movies and 6040 users.



**Fig. 11** The path model constructed for EX1 - Choice overload experiment. Please refer to Sect. 4.2.4 “Graphical presentation of SEMs” for interpretation of this figure. Note that the manipulation of recommendation set size and quality is represented by the two rectangles “Top-20” (a larger set than Top-5) and “Lin-20” (a larger set than Top-5, but with lower ranked additional items)

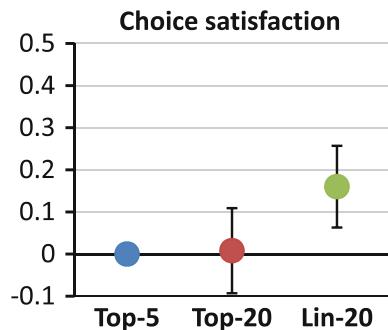
→ outcome-EXP) and a negative effect of choice difficulty (process-EXP → outcome-EXP). Furthermore, high-quality recommendations increase choice difficulty (SSA → process-EXP), and recommendation quality itself depends strongly on the perceived recommendation variety (SSA → SSA; consistent with FT3 and FT4).

Relative to Top-5, the Top-20 condition increases the perceived variety of the recommendation set (OSA → SSA) and the difficulty of the choice (OSA → process-EXP): Top20 is more varied and more difficult to choose from. The Lin-20 set is also more varied than Top-5 (OSA → SSA), and negatively affects the recommendation set attractiveness (OSA → SSA). Lin-20 furthermore has a positive residual effect (i.e. controlling for recommendation set quality and choice difficulty) on satisfaction (OSA → outcome-EXP), relative to the Top-5 condition. Finally, in contrast to the findings of [Kamis and Davern \(2004\)](#) and [Hu and Pu \(2010\)](#), increased movie expertise in our study positively affects recommendation set attractiveness and perceived variety (PC → SSA).

#### 4.7.3 Discussion of results

The marginal effect of our manipulation of the recommendation set composition on choice satisfaction is zero (requirement 2, see Sect. 3.7); there is no significant marginal difference between the Top-5, Top-20 and Lin-20 sets (see Fig. 12). Thanks to our mediating SSAs and EXPs (requirement 4 and 5), the model is able to show exactly why this is the case. The Top-20 set is more varied (SSA) than Top-5 (and thereby more attractive), but it is also more difficult (process-EXP) to make a choice from this set, and these effects level out to eventually show no difference in choice

**Fig. 12** Tendency of marginal (direct and indirect) effects of the “recommendation set composition” condition on choice satisfaction. Error bars indicate  $\pm 1$  Standard Error. The value of Top-5 is fixed to zero; scales are in sample standard deviations



satisfaction (outcome-EXP). The Lin-20 set is less attractive than the Top-5 (SSA), but there is a positive residual effect on choice satisfaction (outcome-EXP). Arguably, this can be interpreted as a context effect: participants in this condition contrasted their choice against the inferior items that are in the tail of the Lin-20 distribution, making the choice more satisfying because it was easier to justify. In the end, however, these effects too cancel out, so the resulting net effect on choice satisfaction is approximately zero.

Finally, the results show that domain knowledge (PC) influences the perception of recommendation set variety and quality. Although requirement 8 specifies that PC should be related to EXP or INT, we have now at several occasions shown a relationship between PC/SC and SSA. Apparently, situational and personal characteristics can influence users’ perceptions as well (e.g. “an expert’s eye”), something not recognized in the earlier version of our framework (Knijnenburg et al. 2010a).

#### 4.8 EX2 diversification experiment

As a final experiment, we set out to investigate the relative effect of algorithm accuracy and recommendation set diversity on user experience. Ziegler et al. (2005) already showed that diversifying the output of a recommender algorithm, although detrimental for the accuracy of the recommended list, results in an increase in overall satisfaction (OSA → EXP). However, their experiment used single-question measures and a restricted set of algorithms (only item-based and user-based k-Nearest Neighbors). Furthermore, as we have shown in our previous experiments, perceived variety and perceived quality of the recommendations (SSAs) are important mediators between objective aspects (OSA) and user experience (EXP).

To better understand the intricate relation between accuracy, diversity and satisfaction, we therefore conducted an experiment in which we manipulated algorithm and recommendation set variety (OSA) and measured perceived accuracy and diversity (SSAs) as well as several user experience variables (process-EXP, system-EXP and outcome-EXP). The experiment compares k-Nearest Neighbors (kNN) with a non-personalized “generally most popular” (GMP) algorithm (a baseline) and a state-of-the-art Matrix Factorization algorithm (MF). Like EX1, it uses the MovieLens 1M dataset.

#### 4.8.1 Setup

The experiment was conducted online, using Amazon's Mechanical Turk service to gather 137 participants (78 male, mean age of 30.2,  $SD = 10.0$ ), mainly from the US and India. Participants were paid US \$4 upon successful completion of the experiment. They were asked to rate at least ten items from the MovieLens 1M set, after which they would receive a list of ten recommendations. The composition of the list was manipulated in  $3 \times 3$  conditions, independently varying the algorithm (GMP, kNN, MF) and the diversification (none, little, lot) between subjects. Participants were asked to choose one movie. Finally, they filled out the user experience questionnaires.

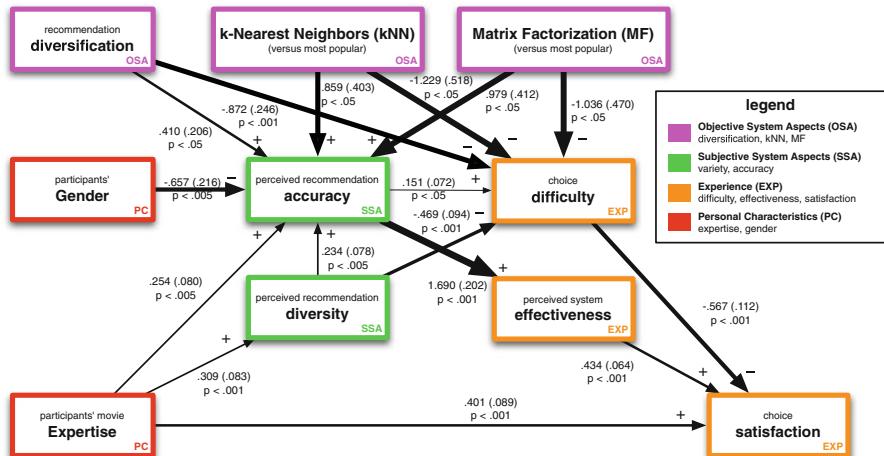
Diversification was based on movie genre, and implemented using the greedy heuristic of Ziegler et al. (2005). Specifically, the heuristic starts with the item with the highest predicted rating, calculates a diversity score for each of the remaining items in the top 100 (based on how many movies in the current recommendation list were of the same genre), creates a compound score  $S$  weighing the predicted rating  $R$  and the diversity score  $D$  according to the formula  $S = aD + (1 - a)R$ , and then chooses from the top 100 the item with the highest compound score. This process is repeated until the list contains ten items. For none, little and lot, the values of 'a' were 0, 0.5 and 0.9 respectively. Diversity scores were calculated using cosine similarity between the genres of the movies.

Participants answered 40 questionnaire items on a 7-point scale. The answers were factor analyzed and produced 6 conceptual factors:

- Perceived recommendation set diversity (5 items, e.g. "Several movies in the list of recommended movies were very different from each other", factor  $R^2 = .087$ )
- Perceived recommendation set accuracy<sup>16</sup> (6 items, e.g. "The recommended movies fitted my preferences", factor  $R^2 = .256$ )
- Perceived choice difficulty (4 items, e.g. "Selecting the best movie was easy/difficult", factor  $R^2 = .312$ )
- Perceived system effectiveness (7 items, e.g. "The recommender system gives me valuable recommendations", factor  $R^2 = .793$ )
- Choice satisfaction (6 items, e.g. "My chosen movie could become one of my favorites", factor  $R^2 = .647$ )
- Expertise (3 items, e.g. "Compared to my peers I watch a lot of movies", no incoming arrows)

8 items were deleted due to low communalities or excessive cross-loadings. The discovered constructs were then causally related with each other and with the  $3 \times 3$  conditions in a Structural Equation Model. The resulting model (Fig. 13) has a reasonably good fit ( $\chi^2(68) = 132.19$ ,  $p < .001$ ,  $CFI = .954$ ,  $TLI = .976$ ,  $rMSEA = .083$ ). The interactions between diversification and algorithm were included in the model, but not in the graph. This means that the effect of diversification in the graph holds

<sup>16</sup> In the previous studies we used the concept "perceived recommendation quality" instead of "perceived recommendation accuracy". The concepts of "recommendation quality" and "recommendation accuracy" are slightly different: perceived accuracy merely looks at how well the recommendations fit ones preferences, while recommendation quality allows for other possible sources of quality.



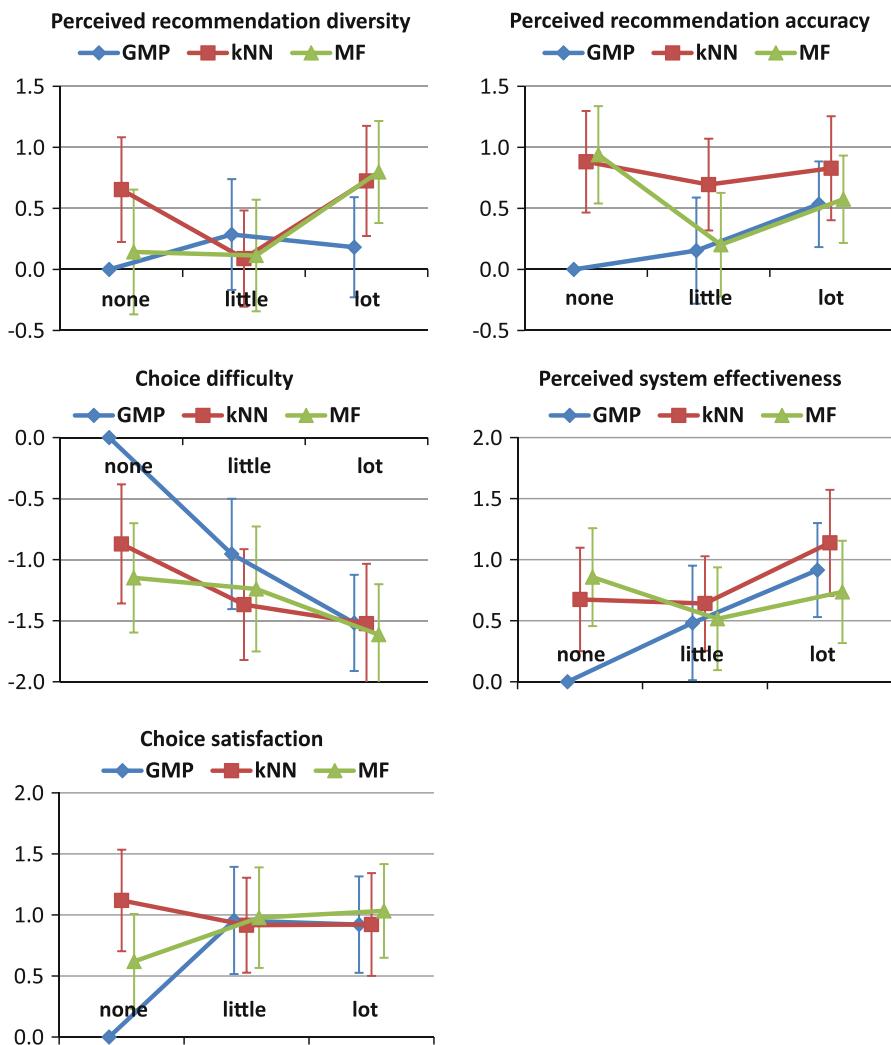
**Fig. 13** The path model constructed for EX2 - Diversification experiment. Please refer to Sect. 4.2.4 “Graphical presentation of SEMs” for interpretation of this figure. The manipulation “diversification” gives values 0, 1, and 2 to levels “none”, “little” and “lot” respectively. The manipulation of algorithm is represented by the two rectangles “k-Nearest Neighbors” and “Matrix Factorization”, each tested against the “generally most popular” recommendations. Figure 14 gives a more insightful presentation of the effects of our conditions

only for the “generally most popular” algorithm condition, and that the effects of kNN and MF hold only for the non-diversified condition.

#### 4.8.2 Results

The model shows that non-diversified recommendations provided by the kNN and MF algorithms are perceived as more accurate than the non-diversified “generally most popular” (GMP) recommendations (OSA → SSA). Diversified GMP recommendations are also perceived as more accurate than non-diversified GMP recommendations (OSA → SSA), even though their predicted rating has been traded off with diversity (i.e., the predicted rating of the diversified item set is lower). Accurate recommendations are generally more difficult to choose from (SSA → process-EXP), however, there is also a direct negative effect of kNN, MF and diversification on choice difficulty (OSA → process-EXP). Looking at the marginal effects in Fig. 14 (“Choice difficulty”) we observe that it is less difficult to choose from the recommendations produced by the high diversification settings and the kNN and MF algorithms (or in other words: it is mainly the non-diversified generally most popular recommendations that are difficult to choose from).

Surprisingly, the lack of a direct effect of diversification on perceived diversity suggests that diversified recommendations do *not* seem to be perceived as more diverse. Figure 14 (“Perceived recommendation diversity”) gives a possible explanation for this lack of effect. The relation between manipulated diversity and perceived diversity is not consistent between the three algorithms. Most notably, two observations differ strongly from our initial expectations: the kNN algorithm with low diversification



**Fig. 14** Tendency of marginal (direct and indirect) effects of algorithm and diversification on our subjective constructs. Error bars indicate  $\pm 1$  Standard Error. The value of GMP-none is fixed to zero; scales are in sample standard deviations

provides surprisingly diverse recommendations, and the most diversified “generally most popular” recommendations are not perceived to be as diverse as they should be.

In general, more diverse recommendations (in this case *perceived* diversity) are also perceived as more accurate (SSA → SSA) and less difficult to choose from (SSA → process-EXP; see also Willemse et al. 2011). Users who rate the recommendations as accurate also perceive the system to be more effective (SSA → system-EXP). The more effective the system and the easier the choice, the more satisfied participants are

with their choice (process-EXP → outcome-EXP and system-EXP → outcome-EXP, respectively).

Finally, we observe that males find the recommendations generally less accurate (PC → SSA), and that in contrast to findings by [Kamis and Davern \(2004\)](#) and [Hu and Pu \(2010\)](#) expertise increases the perceived accuracy and diversity of the recommendations (PC → SSA) and also increases the choice satisfaction (PC → outcome-EXP).

#### 4.8.3 Discussion of results

Like [Ziegler et al. \(2005\)](#), our results show a positive effect of diversification on the user experience (requirement 2, see Sect. 3.7). The inclusion of several user experience constructs in our experiment enables us to show that this effect is mainly due to a lower choice difficulty and a higher perceived system effectiveness (requirement 5). Interestingly, this effect is partially mediated by perceived accuracy, but *not* by perceived diversity (requirement 4). Arguably, users do not necessarily see the diversified recommendations as more diverse. The higher perceived accuracy may have resulted from the fact that users evaluate the accuracy in our questionnaire not per item, but for the list of recommendations as a whole. Users may have noticed that more diverse lists more accurately reflect their diverse tastes.

Figure 14 shows that the “GMP-none” condition (i.e. the non-diversified condition without diversification) stands out; it is the only condition that results in a significantly worse user experience. Also, Fig. 13 shows that the effects of diversification and algorithm are similar: they both increase recommendation accuracy and decrease choice difficulty. In other words: diversification of GMP recommendations might be just as effective in improving the user experience as introducing a recommendation algorithm. Moreover, despite the absence of an interaction effect in the model, Fig. 14 suggests that these effects are not additive: introducing diversification *and* personalization does not improve the user experience beyond introducing either of them separately. Possibly, there is a ceiling-effect to the improvement of the perceived recommendation quality, at least in the between subjects design we employed in this study in which participants do not compare between conditions themselves directly. The result seems to suggest that providing a diversified but non-personalized list of recommendations may result in an equally good user experience as providing personalized recommendations, but we would suggest a replication with other datasets to confirm this surprising observation.

We are puzzled by the fact that our diversification manipulation did not result in a higher perceived diversity of the recommendations. Our diversification algorithm reduces the similarity between movies in terms of genre. This may not fit the definition of similarity as the users of the system judge it. Alternatively, our measurement of this concept could have been too weak to pick up on the subtle differences between the recommendation lists. Finally, the effects of expertise and gender show support for requirement 8: users’ personal characteristics do indeed influence their user experience.

## 5 Validation of the framework

The goal of this paper is to describe a generic framework for the user-centric evaluation of recommender systems. Such a framework should have merits beyond the scope of a single research project. Despite their limited scopes, the field trials and experiments described in this paper can provide an initial test of the general applicability of the framework; any (lack of) consistency in our results gives valuable cues concerning the external validity of the framework. Furthermore, the framework was designed to allow for ad-hoc findings that can be elaborated in future work. Taken together, the expected and unexpected findings of our field trials and experiments allow us to reflect on the merit of our framework as a guiding tool for user-centric recommender systems research.

Below, we consolidate the findings of our field trials and experiments by validating parts of the framework separately and discussing the merit of the framework as a whole. We also assess whether parts of the framework need to be reconsidered in future investigations.

The findings are summarized in Table 2. The table shows that most requirements (as defined in Sect. 3.7) are covered by at least two studies. As the requirements arise from gaps in the existing literature, the results of our studies bridge these gaps, and at the same time provide a thorough validation of the framework itself. Moreover, most of the specific results are confirmed in more than one study, thereby providing evidence for the robustness of the results as well as the general applicability of the framework.

Based on the current findings and the relation of the framework to a broad range of existing literature, we believe that the general requirements are extensible beyond the scope of the current studies. However, specific results may very well only hold for collaborative filtering media recommenders. Further research is needed to extend these specific findings to other types of recommender systems.

### 5.1 Objective system aspects (OSA) and subjective system aspects (SSA)

We defined Objective System Aspects (OSAs) as the features of a recommender system that may influence the user experience, and as things to manipulate in our studies. Most work in the field of recommender systems is focused on algorithms. In our studies we therefore considered algorithms as a manipulation (requirement 1), but we also considered the type of preference input data (requirement 2) and the composition of the recommendation list (requirement 3). We reasoned that users would notice a change in an OSA: these subjective observations are the Subjective System Aspects (SSAs). Specifically, we argued that SSAs would mediate the effect of the OSAs on the user's experience (EXP; requirement 4).

Taken together, the manipulations in FT1, FT2, FT4 and EX2 show that users are able to perceive higher quality recommendations (OSA → SSA) and that this perception mediated the effect of the recommendation quality on the user experience (SSA → EXP), even though this mediation is in some cases only partial (i.e. in FT2, EX1, EX2).

**Table 2** A summary of the results of our field trials and experiments, listed per requirement (as defined in Sect. 3.7)

| Requirement                                 | Results   | Studies            |
|---|---|--------------------|
| 1. Algorithm                                | Turning recommendations on or off has a noticeable effect on user experience  | FT1, FT2, FT4, EX2 |
|   | Experience differences between algorithms are less pronounced   | EX2                |
| 2. Recommendation set composition           | Recommendation set size, quality and diversity have a significant impact on the user experience   | EX1, EX2           |
|   | Sets of different sizes and quality may end up having the same choice satisfaction due to choice overload   | EX1                |
|   | Users do not perceive diversified recommendation sets as more diverse, but they do perceive them as are more accurate   | EX2                |
| 3. Preference input data                    | Explicit feedback leads to more diverse recommendations, which subsequently leads to increased perceived quality; implicit feedback increases the perceived recommendation quality directly | FT4                |
| 4. Perceived aspects as mediators           | Perceived aspects, particularly perceived recommendation quality and variety, provide a better understanding of the results   | All                |
| 5. User experience evaluation               | Usage effort and choice difficulty are measured as process-related experience   | FT2, EX1, EX2      |
|   | Perceived system effectiveness is measured as system-related experience   | All except EX1     |
|   | Choice satisfaction is measured as outcome-related experience   | FT1, FT2, EX1, EX2 |
|   | Process-related experience causes system- or outcome-related experience   | FT2, EX1, EX2      |
| 6. Providing feedback                       | System-related experience causes outcome-related experience   | FT2, EX2           |
|   | Intention to provide feedback is a trade-off between trust/privacy and experience   | FT1, FT2, FT4      |
|   | Users' feedback behavior may not always be correlated with their intentions to provide feedback   | FT1, FT2           |
| 7. Behavioral data                          | A positive personalized user experience is characterized by reduced browsing behavior and increased consumption   | FT1, FT2           |
| 8. Personal and situational characteristics | Users' experience and behaviors change over time  | FT2, FT3           |
|   | Age, gender and domain knowledge have an influence on users' perceptions, experience and behaviors  | FT4, EX1, EX2      |

EX1 and EX2 show similar results for the recommendation set composition. Size, quality and diversification of the recommendation set each influence the user experience via subjective perceptions (OSA → SSA → EXP), even though the effect on choice difficulty is mainly direct. Surprisingly, our diversification algorithm increased

perceived accuracy but not perceived diversity. Despite this, diversification was as effective in improving the user experience as the introduction of a good algorithm.

In terms of preference input, FT4 shows that even though implicit feedback recommendations are based on more data than explicit feedback recommendations (all user behavior versus ratings only), they are not always better from a user's perspective, especially if one takes into account the variety of the recommendations.

Concluding, the core component of our framework—the link from algorithm or preference input data (OSA) to subjective recommendation quality (SSA) to experience (EXP)—is upheld throughout every conducted study. Some direct effects (OSA → EXP) occur, which could indicate that not all possible SSAs were measured. In general the SSAs mediate the effects of OSA on EXP, thereby explaining the effects of the OSAs in more detail (i.e. the why and how of improved user experience). The capability to explain both surprising and straightforward findings makes SSAs an essential part of our evaluation.

## 5.2 The different objects of experience (EXP)

We reasoned that user experience could be process-related, system-related and outcome-related. We argued that different recommender system aspects could influence different types of experience, and that one type of experience could influence another. Previous research typically includes just one type of user experience in their evaluation. In most of our studies we therefore considered more than one type of experience (requirement 5).

Our research confirms that these different types of user experience indeed exist; the studies discern the following user experience constructs: perceived usage effort (process-EXP; FT2), choice difficulty (process-EXP; EX1, EX2), perceived system effectiveness (system-EXP; all studies, except EX1) and choice satisfaction (outcome-EXP; FT1, FT2, EX1, EX2). In all cases, the main mediator that causes these experience variables is the perceived recommendation quality. Structurally, process-related experiences often cause system-related experiences, which in turn cause outcome-related experiences (process-EXP → system-EXP → outcome-EXP).

## 5.3 Behavioral data and feedback intentions

We indicated that behavioral data could be triangulated with subjective experience data to improve the interpretation of behavioral results, and to ground self-report measures in actual behavior (requirement 7). We also reasoned that the specific behavior of providing preference feedback is of particular interest to recommender systems researchers, because in many systems this feedback is needed to provide good recommendations (requirement 6).

FT1, FT2 and FT4 show that feedback behavior is a trade-off between the users' trust or privacy concerns, and the user experience. The actual feedback behavior is not always highly correlated with the intention to provide feedback.

Furthermore, all field trials show several significant triangulations. Consistently, reduced browsing and increased consumption are indicators of effective systems.

In some cases behaviors are directly related to the OSAs. This probably means that we did not measure the specific SSAs that would mediate the effect.

#### 5.4 Personal and situational characteristics (PC and SC)

We argued that personal and situational characteristics may influence the users' experience and interaction with the system (requirement 8). Our research (FT1 and FT2) addresses trust in technology (PC) and system-specific privacy concerns (SC), and shows how these concepts influence users' feedback intentions (see requirement 6). Furthermore, the experience and interaction of the recommender systems in our research changes over time (SC; see FT2 and FT3). Concluding, several contextual effects seem to influence the users' interaction and experience with the recommender system. Our research addresses trust, privacy, time, age, gender and expertise (domain knowledge).

Our initial conception of the effect of personal and situational characteristics seems to have been too restrictive: We only allowed these characteristics to influence the experience (EXP) and interaction (INT). Based on the results of our research, we acknowledge that these characteristics can sometimes influence not only the evaluation, but also the perception of the system (i.e. influence the subjective system aspects, SSAs). We suggest including an additional arrow from personal and situational characteristics to subjective system aspects ( $PC \rightarrow SSA$  and  $SC \rightarrow SSA$ ), and we encourage researchers to investigate this connection in more detail in future experiments.

Our research on the users' intention to provide feedback also alludes to possible changes in the framework. The results of FT1 and FT2 are not entirely consistent in terms of the factors that influence the intention to provide feedback, and further research is needed to specifically delineate what causes and inhibits users to provide preference feedback to the system (see [Pommeranz et al. 2012](#), for a good example).

### 6 Conclusion

The framework provides clear guidance for the construction and analysis of new recommender system experiments. It allows for an in-depth analysis that goes beyond algorithmic performance: it can explain *why* users like a certain recommender system and *how* this user experience comes about.

The framework also puts emphasis on the integration of research. When testing with real users one cannot study algorithms in isolation; several system aspects (and personal and situational characteristics) have to be combined in a single experiment to gain a full understanding of the user experience.

For industry researchers, the user-centric focus of the framework provides a step closer to the customers, who may not consider the accuracy of the algorithm the most important aspect of their experience. Questionnaire-taking and A/B testing (the industry term for testing several versions of a certain system aspect) are an accepted form of research in web technology. For academic researchers, the framework provides an opportunity to check the real-world impact of the latest algorithmic improvements. Moreover, interesting effects of situational and personal characteristics, as well as

behavioral correlates can be used as input for context-aware recommender engines. Even more so, when evaluating the relative merit of novel recommendation approaches such as context-aware algorithms (Adomavicius et al. 2005) and recommenders using social networks (Kautz et al. 1997), one has to rely on more sophisticated ways of measuring the full user experience, and our framework could serve as a guideline for such evaluations.

## 7 Future research

This paper has argued that measuring algorithmic accuracy is an insufficient method to analyze the user experience of recommender systems. We have therefore introduced and validated a user-centric evaluation framework that explains how and why the user experience of a recommender system comes about. With its mediating variables and its integrative approach, the framework provides a structurally inclusive foundation for future work. Our research validates the framework and, beyond that, produced some unanticipated results.

Still, our work represents merely the tip of the iceberg of user-centric recommender systems research. Our research is limited in scope: we only tested a small number of media-oriented recommender systems. To determine the scope of applicability of our framework, further validation of the framework should consider other content types, specifically “search products”. Moreover, some of our results are inconclusive and require further investigation.

The link between algorithmic accuracy and user experience is a fundamental question that currently still functions as the untested premise of a considerable part of the recommender systems research. The same holds for users’ intention to provide feedback. And whereas our research shows some interesting results concerning the use of explicit versus implicit feedback, it is by no means exhaustive in this respect. We furthermore believe that future work could investigate the effects of other personal and situational characteristics, and the results of these studies could be used to personalize not only the recommendations of the system, but also the system itself (Knijnenburg and Willemsen 2010, 2009; Knijnenburg et al. 2011).

Because of the pioneering nature of our work, we took a very thorough approach in our evaluation. The proposed methodology of measuring constructs with multiple questionnaire items and analyzing the results with exploratory factor analyses and structural equation models improves the external validity of our results. We realize, however, that this methodology may be infeasible when testing recommender systems in a fully operational industry setting. We therefore created a pragmatic procedure that allows the measurement of specific concepts of recommender system user experience with just a few key questionnaire items and a simplified (possibly automated) statistical evaluation (Knijnenburg et al. 2011a). Our current results provided useful input for the development of this procedure.

Concluding, our framework provides a platform for future work on recommender systems, and allows researchers and developers in industry and academia to consistently evaluate the user experience of their systems. The future of user-centric recommender systems research is full of exciting opportunities.

**Acknowledgements** We would like to thank Mark Graus for programming the recommender systems used in EX1 and EX2, Steffen Rendle for implementing the explicit feedback MF algorithm, Niels Reijmer, Yunan Chen and Alfred Kobsa for their comments at several stages of this paper, and Dirk Bollen for allowing us to incorporate the results of his choice overload experiment (EX1) in this paper. We also thank the three anonymous reviewers for their extensive comments on the initial submission. We gratefully acknowledge the funding of our work through the European Commission FP7 project MyMedia ([www.mymediaproject.org](http://www.mymediaproject.org)) under the grant agreement no. 215006. For inquiries please contact info@mymediaproject.org.

**Open Access** This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

## Appendix A: Methodology

Statistical analysis of our field trials and experiments involved two main steps: validating the measured latent concepts using exploratory factor analysis (EFA) and testing the structural relations between the manipulations, latent concepts and behavioral measurements using structural equation modeling (SEM). In this section we provide a detailed description of our methodological approach by means of a hypothetical example. See [Knijnenburg et al. \(2011a\)](#) for a more pragmatic procedure to evaluate recommender systems.

In the example, we test two algorithms ( $A_1$  and  $A_2$ ) against a non-personalized baseline ( $A_0$ ). We measure perceived recommendation quality (Q) with 5 statements ( $Q_1 \dots Q_5$ ) to which participants can agree or disagree on a 5-point scale. Satisfaction with the system (S) is measured with 6 items ( $S_1 \dots S_6$ ). Finally, we measure user behavior (B) in terms of the number of clips watched from beginning to end.

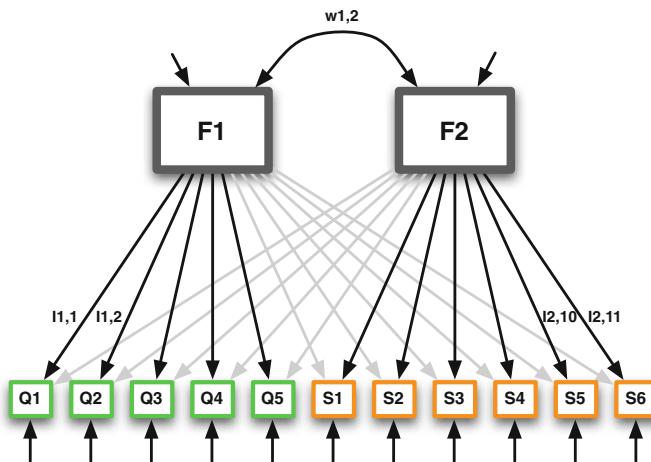
### Exploratory factor analysis (EFA)

The first step is to confirm whether the 13 items indeed measure the predicted two latent concepts. This is done using exploratory factor analysis. This technique extracts common variance between the measured items and distributes this variance over a number of latent factors. We use the software package Mplus<sup>17</sup> to do this analysis with the command “analysis: type = efa 1 3; estimator = wlsmv;”. This runs the exploratory factor analysis with 1, 2 and 3 factors, and uses a weighted least squares estimation procedure with mean- and variance-adjusted chi-square tests.

The factor analytical model can be represented as Fig. 15. Each item is essentially a regression outcome,<sup>18</sup> predicted by two unobserved latent variables.  $I_{1,1}$  to  $I_{2,11}$  are called the loadings of the items  $Q_1 \dots S_6$  on the factors  $F_1$  and  $F_2$ . The model tries to estimate these loadings so that the paths match the covariance matrix of the items  $Q_1$  to  $S_6$  as closely as possible (e.g.  $\text{cov}_{1,2} \approx I_{1,1} * I_{1,2} + I_{2,1} * I_{2,2} + I_{1,1} * w_{1,2} * I_{2,2} + I_{1,2} * w_{1,2} * I_{2,1}$ ). Intuitively, factor analysis tries to model the “overlap” between items. The part of the variance that does not overlap is excluded (and represented by

<sup>17</sup> <http://www.statmodel.com/>.

<sup>18</sup> Since these outcomes are measured on a 5-point scale, this regression model is an ordinal response model.



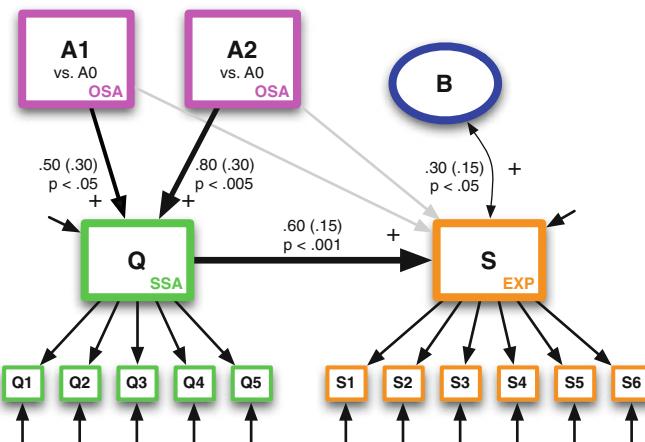
**Fig. 15** Representation of the exploratory factor analysis example. Two factors,  $F_1$  and  $F_2$ , are extracted from the questionnaire data (items  $Q_1 \dots Q_5$  and  $S_1 \dots S_6$ ). The arrows from the factors to the questions ( $I_{1,1} \dots I_{2,11}$ ) represent the factor loadings. The arrows at the bottom represent the portion of the variance not accounted for by the analysis. Factors have a certain reliability (represented by the arrows at the top), and may be correlated (represented by  $w_{1,2}$ ). If the grey arrows are close to zero,  $F_1$  effectively measures recommendation quality (Q) and  $F_2$  effectively measures satisfaction with the system (S)

the arrows at the bottom of Fig. 15). The more overlap extracted, the more reliable the factor (the arrows at the top). The factors may be correlated with each other ( $w_{1,2}$ ). The solution has no standard coordinate system, so it is rotated so that each question loads on only one factor as much as possible.

If the measurement tool was specified correctly, then the model has a good least-squares fit, and only the paths from  $F_1$  to  $Q_1 \dots Q_5$  and from  $F_2$  to  $S_1 \dots S_6$  are significantly larger than zero (i.e. only the darker paths in Fig. 15). In that case  $F_1$  measures recommendation quality, and  $F_2$  measures satisfaction with the system. However, the following problems may arise:

- A one-factor solution fits almost as well as a two-factor solution: In this case, we must conclude that Q and S are essentially the same concept.
- A three-factor solution fits much better than a two-factor solution: In this case, this questionnaire measures three concepts (usually because either S or Q is in fact a combination of two factors).
- A certain item does not have enough in common with the other items to load on any of the factors significantly ( $p < .05$ ): In this case the item has “low communality” and should be removed from the analysis.
- A certain item loads on the wrong factor, or on both factors: The item has a high “cross-loading”, and unless we can come up with a good reason for this to occur, it should be removed from the analysis.

Once an adequate factor solution is established, the remaining items are used in the second step of the analysis.



**Fig. 16** Representation of the structural equation modeling example. The algorithms ( $A_1$  and  $A_2$ ) influence the perceived recommendation quality ( $Q$ ), which in turn influences the satisfaction with the system ( $S$ ). The satisfaction ( $S$ ) is in turn correlated with the number of clips watched from beginning to end ( $B$ ).  $Q$  is measured by ( $Q_1 \dots Q_5$ ) and  $S$  is measured by ( $S_1 \dots S_6$ ). In the models in the main text the questionnaire items ( $Q_1 \dots S_6$ ) are hidden in order to get a less cluttered representation

### Structural equation modeling (SEM)

The second step is to test the structural relations between our manipulation (A), the latent concepts (Q and S) and the behavior measurement (B). Our manipulation has three conditions ( $A_0$ ,  $A_1$ , and  $A_2$ ), so we create two dummy variables ( $A_1$  and  $A_2$ ), which are tested against the baseline. The dummies are coded in such a way that they represent the different conditions: For participants in the baseline  $A_1 = 0$ , and  $A_2 = 0$ ; for participants with algorithm 1,  $A_1 = 1$ , and  $A_2 = 0$ ; for participants with algorithm 1,  $A_1 = 0$ , and  $A_2 = 1$ . As a result of this coding, the effect of  $A_1$  is therefore the effect of algorithm 1 compared to the baseline, and the effect of  $A_2$  is the effect of algorithm 2 compared to the baseline.

The resulting model is tested using structural equation modeling. The specification of the model defines the factors and the items with which they are measured (in Mplus: “Q by  $Q_1-Q_5$ ; S by  $S_1-S_6$ ;”), and the structural relations among the factors and other variables (in Mplus: “S on Q  $A_1 A_2$ ; Q on  $A_1 A_2$ ;”). This creates a model that can be represented as Fig. 16 (but, as of yet, without B).

Like any regression model, structural equation models make assumptions about the direction of causality in the model. From a modeling perspective, an effect ( $Q \rightarrow S$ ) and its reverse ( $S \rightarrow Q$ ) are equally plausible (Netemeyer and Bentler 2001). By including the manipulation(s) in our model, we are able to “ground” the causal effects: participants are randomly assigned to a condition, so condition assignment cannot be caused by anything in the model (i.e.  $A_1 \rightarrow Q$  is possible, but not  $Q \rightarrow A_1$ ). Furthermore, the framework provides hypotheses for the directionality of causal effects (since in the framework we hypothesize that  $SSA \rightarrow EXP$  and not  $EXP \rightarrow SSA$ , we can limit ourselves to testing  $Q \rightarrow S$ , cf. Anderson and Gerbing 1988; Bagozzi and Yi 1988).

For each regression path in the model, a regression coefficient is estimated. As the values of the latent constructs are standardized (by means of the factor analysis), the regression coefficient between the two latent constructs ( $Q \rightarrow S$ ) shows that a 1 standard deviation difference in  $Q$  causes a 0.60 standard deviation difference  $S$ . The standard error of the coefficient (0.15) can be used in a z-test to test whether the path is significant ( $p = z[.60/.15] = .00003 < .001$ ). The dummy variables  $A_1$  and  $A_2$  are not standardized, but represent the presence (1) or absence (0) of a certain condition. Therefore, the coefficient on the arrow  $A_1 \rightarrow Q$  shows that  $Q$  is 0.50 standard deviations higher for participants in  $A_1$  than for those in  $A_0$ .

In a typical model, not all initially specified paths are significant. This means that some effects will be fully mediated. For example: if the paths from  $A_1$  and  $A_2$  to  $S$  (the lighter paths in Fig. 16) are not significant, the effect of  $A_1$  and  $A_2$  on  $S$  is fully mediated. Otherwise, there is only partial mediation. Non-significant effects are removed from the model and the model is ran again to create a more parsimonious result.<sup>19</sup>

Variables that have no hypothesized effect (such as  $B$  in this model) are initially included in the model without specifying any structural relation for it. Mplus can provide a “modification index” for this variable, thereby showing where it best fits in the model. Since such effect is ad-hoc, it is only to be included if it is highly significant.

Variables for which we have no hypothesized direction of effect (again, variable  $B$  in this model, for which we don’t know whether  $B \rightarrow S$  or  $S \rightarrow B$ ) are included as a correlation. This means that no assumption about the direction of causality is made.

The final model (after excluding non-significant effects and including ad-hoc effects) can be tested as a whole. The Chi-square statistic tests the difference in explained variance between the proposed model and a fully specified model. A good model is not statistically different from the fully specified model ( $p > .05$ ). However, this statistic is commonly regarded as too sensitive, and researchers have therefore proposed other fit indices (Bentler and Bonett 1980). Based on extensive simulations, Hu and Bentler (1999) propose cut-off values for these fit indices to be:  $CFI > .96$ ,  $TLI > .95$ , and  $rMSEA < .05$ . Moreover, a good model makes sense from a theoretical perspective. The model shown in Fig. 16 has this quality: The algorithms ( $A_1$  and  $A_2$ ) influence the perceived recommendation quality ( $Q$ ), which in turn influences the satisfaction with the system ( $S$ ). The satisfaction ( $S$ ) is in turn correlated with the number of clips watched from beginning to end ( $B$ ).

## Appendix B: Questionnaire items for each construct

This appendix lists all the questionnaire items shown to the participants of the field trials and experiments. It makes a distinction between those items that were included

<sup>19</sup> Not all non-significant effects are excluded from the models. For instance, when two conditions are tested against a baseline, and one of the conditions does not significantly differ from the baseline but the other does, then the non-significant effect is retained to allow a valid interpretation of the significant effect. Furthermore, in EX2, the interactions between diversity and algorithm, though not significant, are retained, as they allow a valid interpretation of the significant conditional main effects.

in the final analysis, and those items that did not contribute to stable constructs and were therefore deleted. Included questions are in order of decreasing factor loading.

### FT1 EMIC pre-trial

#### *Perceived recommendation quality*

Included:

- I liked the items recommended by the system.
- The recommended items fitted my preference.
- The recommended items were well-chosen.
- The recommended items were relevant.
- The system recommended too many bad items.
- I didn't like any of the recommended items.
- The items I selected were “the best among the worst”.

#### *Perceived system effectiveness*

Included:

- I would recommend the system to others.
- The system is useless.
- The system makes me more aware of my choice options.
- I make better choices with the system.
- I can find better items without the help of the system.
- I can find better items using the recommender system.

Not included:

- The system showed useful items.

#### *Choice satisfaction*

Included:

- I like the items I've chosen.
- I was excited about my chosen items.
- I enjoyed watching my chosen items.
- The items I watched were a waste of my time.
- The chosen items fit my preference.
- I know several items that are better than the ones I selected.
- Some of my chosen items could become part of my favorites.
- I would recommend some of the chosen items to others/friends.

#### *Intention to provide feedback*

Included:

- I like to give feedback on the items I'm watching.
- Normally I wouldn't rate any items.
- I only sparingly give feedback.

- I didn't mind rating items.
- In total, rating items is not beneficial for me.

### *General trust in technology*

Included:

- Technology never works.
- I'm less confident when I use technology.
- The usefulness of technology is highly overrated.
- Technology may cause harm to people.

Not included:

- I prefer to do things by hand.
- I have no problem trusting my life to technology.
- I always double-check computer results.

### *System-specific privacy concern*

Included:

- I'm afraid the system discloses private information about me.
- The system invades my privacy.
- I feel confident that the system respects my privacy.
- I'm uncomfortable providing private data to the system.
- I think the system respects the confidentiality of my data.

### FT2 EMIC trial

#### *Perceived recommendation quality*

Included:

- I liked the items shown by the system.
- The shown items fitted my preference.
- The shown items were well-chosen.
- The shown items were relevant.
- The system showed too many bad items.
- I didn't like any of the shown items.

Not included:

- The system showed useful items.
- The items I selected were “the best among the worst”.

#### *Effort to use the system*

Included:

- The system is convenient.
- I have to invest a lot of effort in the system.
- It takes many mouse-clicks to use the system.

Included:

- Using the system takes little time.
- It takes too much time before the system provides adequate recommendations.

#### *Perceived system effectiveness and fun*

Included:

- I have fun when I'm using the system.
- I would recommend the system to others.
- Using the system is a pleasant experience.
- The system is useless.
- Using the system is invigorating.
- The system makes me more aware of my choice options.
- Using the system makes me happy.
- I make better choices with the system.
- I use the system to unwind.
- I can find better items using the recommender system.

Not included:

- I can find better items without the help of the system.
- I feel bored when I'm using the system.

#### *Choice satisfaction*

Included:

- I like the items I've chosen.
- I was excited about my chosen items.
- I enjoyed watching my chosen items.
- The items I watched were a waste of my time.
- The chosen items fit my preference.

Not included:

- I know several items that are better than the ones I selected.
- Some of my chosen items could become part of my favorites.
- I would recommend some of the chosen items to others/friends.

#### *Intention to provide feedback*

Included:

- I like to give feedback on the items I'm watching.
- Normally I wouldn't rate any items.
- I only sparingly give feedback.
- I didn't mind rating items.

Not included:

- In total, rating items is not beneficial for me.

*General trust in technology*

Included:

- Technology never works.
- I'm less confident when I use technology.
- The usefulness of technology is highly overrated.
- Technology may cause harm to people.

*System-specific privacy concern*

Included:

- I'm afraid the system discloses private information about me.
- The system invades my privacy.
- I feel confident that the system respects my privacy.

Not included:

- I'm uncomfortable providing private data to the system.
- I think the system respects the confidentiality of my data.

## FT3 BBC pre-trial

*Perceived recommendation quality*

Included:

- The recommended items were relevant.
- I liked the recommendations provided by the system.
- The recommended items fitted my preference.
- The MyMedia recommender is providing good recommendations.
- I didn't like any of the recommended items.
- The MyMedia recommender is not predicting my ratings accurately.
- The recommendations did not include my favorite programmes.

*Perceived recommendation variety*

Included:

- The recommendations contained a lot of variety.
- All the recommended programmes were similar to each other.

*Perceived system effectiveness*

Included:

- The MyMedia recommender is useful.
- I would recommend the MyMedia recommender to others.
- The MyMedia recommender has no real benefit for me.
- I can save time using the MyMedia recommender.
- I can find better programmes without the help of the MyMedia recommender.

- The MyMedia recommender is recommending interesting content I hadn't previously considered.

Not included:

- The system gave too many recommendations.

#### FT4 BBC trial

##### *Perceived recommendation quality*

Included:

- The MyMedia recommender is providing good recommendations.
- I liked the recommendations provided by the system.
- The recommended items fitted my preference.
- The recommended items were relevant.
- I didn't like any of the recommended items.
- The MyMedia recommender is not predicting my ratings accurately.
- The recommendations did not include my favorite programmes.

##### *Perceived recommendation variety*

Included:

- The recommendations contained a lot of variety.
- The MyMedia recommender is recommending interesting content I hadn't previously considered.
- The recommendations covered many programme genres.
- All the recommended programmes were similar to each other.
- Most programmes were from the same genre.

##### *Perceived system effectiveness*

Included:

- The MyMedia recommender has no real benefit for me.
- I would recommend the MyMedia recommender to others.
- The MyMedia recommender is useful.
- I can save time using the MyMedia recommender.
- I can find better programmes without the help of the MyMedia recommender.

Not included:

- The system gave too many recommendations.

#### EX1 choice overload experiment

##### *Perceived recommendation variety*

Included:

- The list of recommendations was varied.
- The list of recommendations included movies of many different genres.

- Many of the movies in the list differed from other movies in the list.
- All recommendations seemed similar.

Not included:

- No two movies in the list seemed alike.
- The list of recommendations was very similar/very varied.

### *Perceived recommendation quality*

Included:

- The list of recommendations was appealing.
- How many of the recommendations would you care to watch?
- The list of recommendations matched my preferences.
- I did not like any of the recommendations in the list.

### *Choice difficulty*

Included:

- Eventually I was in doubt between ... items.
- I changed my mind several times before making a decision.
- I think I chose the best movie from the options.
- The task of making a decision was overwhelming.

Not included:

- How easy/difficult was it to make a decision?
- How frustrating was the decision process?

### *Choice satisfaction*

Included:

- My chosen movie could become one of my favorites.
- How satisfied are you with the chosen movie?
- I would recommend the chosen movie to others.
- I think I would enjoy watching the chosen movie.
- I would rather rent a different movie from the one I chose.
- I think I chose the best movie from the options.

Not included:

- The list of recommendations had at least one movie I liked.

### *Expertise*

Included:

- I am a movie lover.
- Compared to my peers I watch a lot of movies.
- Compared to my peers I am an expert on movies.

## EX2 diversification experiment

### *Perceived recommendation accuracy*

Included:

- The recommended movies fitted my preference.
- Each of the recommended movies was well-chosen.
- I would give the recommended movies a high rating.
- The provided recommended movies were interesting.
- I liked each of the recommended movies provided by the system.
- Each of the recommended movies was relevant.

Not included:

- I did not like any of the recommended movies.

### *Perceived recommendation variety*

Included:

- Several movies in the list of recommended movies were very different from each other.
- The list of recommended movies covered many genres.
- The list of recommended movies had a high variety.
- Most movies were from the same type.
- The list of recommended movies was very similar/ very varied.

Not included:

- All recommended movies were similar to each other.

### *Choice difficulty*

Included:

- Selecting the best movie was very easy/very difficult.
- Comparing the recommended movies was very easy/very difficult.
- Making a choice was overwhelming.
- I changed my mind several times before choosing a movie.

Not included:

- Making a choice was exhausting.
- Making a choice was fun.
- Making a choice was frustrating.
- Eventually I was in doubt between ... movies.

### *Perceived system effectiveness*

Included:

- The recommender system gave me valuable recommendations.
- I would recommend the recommender system to others.
- I can find better movies using the recommender system.

- I make better choices with the recommender system.
- The recommender system is useless.
- The recommender system makes me more aware of my choice options.
- I don't need the recommender system to find good movies.

### *Choice satisfaction*

Included:

- My chosen movie could become one of my favorites.
- The chosen movie fits my preference.
- I will enjoy watching my chosen movie.
- I like the movie I have chosen.
- I will recommend the movie to others/friends.
- Watching my chosen movie will be a waste of my time.

Not included:

- I am excited about my chosen movie.

### *Expertise*

Included:

- Compared to my peers I watch a lot of movies.
- Compared to my peers I am an expert on movies.
- I only know a few movies.

Not included:

- I am a movie lover.

## References

- Ackerman, M., Cranor, L., Reagle, J.: Privacy in e-commerce: examining user scenarios and privacy preferences. In: Conference on Electronic Commerce, pp. 1–8. Denver, CO (1999)
- Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.* **17**, 734–749 (2005)
- Adomavicius, G., Sankaranarayanan, R., Sen, S., Tuzhilin, A.: Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.* **23**, 103–145 (2005)
- Anderson, J.C., Gerbing, D.W.: Structural equation modeling in practice: a review and recommended two-step approach. *Psychol. Bull.* **103**, 411–423 (1988)
- Bagozzi, R., Yi, Y.: On the evaluation of structural equation models. *J. Acad. Market. Sci.* **16**, 74–94 (1988)
- Baudisch, P., Terveen, L.: Interacting with recommender systems. In: SIGCHI Conference on Human Factors in Computing Systems, p. 164. Pittsburgh, PA (1999)
- Bechwati, N., Xia, L.: Do computers sweat? The impact of perceived effort of online decision aids on consumers' satisfaction with the decision process. *J. Consum. Psychol.* **13**, 139–148 (2003)
- Bentler, P.M., Bonett, D.G.: Significance tests and goodness of fit in the analysis of covariance structures. *Psychol. Bull.* **88**, 588–606 (1980)
- Berendt, B., Teltzrow, M.: Addressing users' privacy concerns for improving personalization quality: towards an integration of user studies and algorithm evaluation. In: IJCAI 2003 Workshop on Intelligent Techniques for Web Personalization, LNAI, vol. 3169, pp. 69–88. Acapulco, Mexico (2005)
- Bharati, P., Chaudhury, A.: An empirical investigation of decision-making satisfaction in web-based decision support systems. *Decis. Support Syst.* **37**, 187–197 (2004)

- Bhatnagar, A., Ghose, S.: Online information search termination patterns across product categories and consumer demographics. *J. Retail.* **80**, 221–228 (2004)
- Bollen, D., Knijnenburg, B., Willemsen, M., Graus, M.: Understanding choice overload in recommender systems. In: Fourth ACM Conference on Recommender systems, pp. 63–70. Barcelona, Spain (2010)
- Bradley, K., Smyth, B.: Improving recommendation diversity. In: Twelfth Irish Conference on Artificial Intelligence and Cognitive Science, pp. 85–94. Maynooth, Ireland (2001)
- Brodie, C., Karat, C., Karat, J.: Creating an E-commerce environment where consumers are willing to share personal information. In: Karat, C.-M., Blom, J.O., Karat, J. (eds.) *Designing Personalized User Experiences in eCommerce*, pp. 185–206. Kluwer, Dordrecht (2004)
- Burke, R.: Hybrid recommender systems: survey and experiments. *User Model. User-Adap. Int.* **12**, 331–370 (2002)
- Cena, F., Verner, F., Gena, C.: Towards a customization of rating scales in adaptive systems. In: 18th International Conference on User Modeling, Adaptation, and Personalization, vol. 6075, pp. 369–374. Big Island, HI, LNCS (2010)
- Chellappa, R., Sin, R.: Personalization versus privacy: an empirical examination of the online consumer's dilemma. *Inf. Technol. Manag.* **6**, 181–202 (2005)
- Chen, L., Pu, P.: Interaction design guidelines on critiquing-based recommender systems. *User Model. User-Adap. Inter.* **19**, 167–206 (2009)
- Chen, L., Pu, P.: Critiquing-based recommenders: survey and emerging trends. *User Model. User-Adap. Inter.* **22**(1–2), 125–150 (2012)
- Chin, D.: Empirical evaluation of user models and user-adapted systems. *User Model. User-Adap. Inter.* **11**, 181–194 (2001)
- Cooke, A., Sujan, H., Sujan, M., Weitz, B.A.: Marketing the unfamiliar: the role of context and item-specific information in electronic agent recommendations. *J. Market. Res.* **39**, 488–497 (2002)
- Cosley, D., Lam, S., Albert, I., Konstan, J., Riedl, J.: Is seeing believing?: how recommender system interfaces affect users' opinions. In: SIGCHI Conference on Human Factors in Computing Systems, pp. 585–592. Ft. Lauderdale, FL (2003)
- Cramer, H., Evers, V., van Someren, M., Ramlal, S., Rutledge, L., Stash, N., Aroyo, L., Wielinga, B.: The effects of transparency on trust and acceptance in interaction with a content-based art recommender. *User Model. User-Adap. Inter.* **18**, 455–496 (2008a)
- Cramer, H., Evers, V., van Someren, M., Ramlal, S., Rutledge, L., Stash, N., Aroyo, L., Wielinga, B.: The effects of transparency on perceived and actual competence of a content-based recommender. In: CHI'08 Semantic Web User Interaction Workshop. Florence, Italy (2008b)
- Csikszentmihalyi, M.: Beyond Boredom and Anxiety. Jossey-Bass Publishers, San Francisco (1975)
- Davis, F.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Q.* **13**, 319–340 (1989)
- Davis, F., Bagozzi, R., Warshaw, P.: User acceptance of computer technology: a comparison of two theoretical models. *Manag. Sci.* **35**, 982–1003 (1989)
- Diehl, K., Kornish, L., Lynch, J. Jr.: Smart agents: when lower search costs for quality information increase price sensitivity. *J. Consum. Res.* **30**, 56–71 (2003)
- Felfernig, A., Teppan, E., Gula, B.: Knowledge-based recommender technologies for marketing and sales. *Int. J. Pattern Recog. Artif. Intell.* **21**, 333–354 (2007)
- Felix, D., Niederberger, C., Steiger, P., Stolze, M.: Feature-oriented vs. needs-oriented product access for non-expert online shoppers. In: IFIP Conference on Towards the E-Society: E-commerce, E-business, and E-government, pp. 399–406. Zürich, Switzerland (2001)
- Fishbein, M., Ajzen, I.: Belief, Attitude, Intention and Behavior: An Introduction to Theory and Research. Addison-Wesley, Reading (1975)
- Gena, C., Brogi, R., Cena, F., Verner, F.: Impact of rating scales on user's rating behavior. In: 19th International Conference on User Modeling, Adaptation, and Personalization, LNCS, vol. 6787, pp. 123–134. Girona, Spain (2011)
- Harper F., Li X., Chen Y., Konstan J.: An economic model of user rating in an online recommender system. In: 10th International Conference on User Modeling, LNCS, vol. 3538, pp. 307–316. Edinburgh, UK (2005)
- Hassenzahl, M.: The thing and I: understanding the relationship between user and product. In: Blythe, M.A., Monk, A.F., Overbeeke, K., Wright, P.C. (eds.) *Funology*, pp. 31–42. Kluwer, Dordrecht (2005)

- Hassenzahl, M.: User experience (UX): towards an experiential perspective on product quality. In: 20th International Conference of the Association Francophone d'Interaction Homme-Machine, pp. 11–15. Metz, France (2008)
- Häubl, G., Trifts, V.: Consumer decision making in online shopping environments: the effects of interactive decision aids. *Market. Sci.* **19**, 4–21 (2000)
- Häubl, G., Dellaert, B., Murray, K., Trifts, V.: Buyer behavior in personalized shopping environments. In: Karat, C.-M., Blom, J.O., Karat, J. (eds.) *Designing Personalized User Experiences in eCommerce*, pp. 207–229. Kluwer, Dordrecht (2004)
- Hauser, J., Urban, G., Liberali, G., Braun, M.: Website morphing. *Market. Sci.* **28**, 202–223 (2009)
- Hayes, C., Massa, P., Avesani, P., Cunningham, P.: An on-line evaluation framework for recommender systems. In: AH'2002 Workshop on Recommendation and Personalization in E-Commerce, pp. 50–59. Málaga, Spain (2002)
- Herlocker, J., Konstan, J., Riedl, J.: Explaining Collaborative filtering recommendations. In: 2000 ACM Conference on Computer Supported Cooperative Work, pp. 241–250. Philadelphia, PA (2000)
- Herlocker, J., Konstan, J., Terveen, L., Riedl, J.: Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.* **22**, 5–53 (2004)
- Ho, S.Y., Tam, K.Y.: An empirical examination of the effects of web personalization at different stages of decision making. *Int. J. Human-Comput. Interact.* **19**, 95–112 (2005)
- Hostler, R., Yoon, V., Guimaraes, T.: Assessing the impact of internet agent on end users' performance. *Decis. Supp. Syst.* **41**, 313–323 (2005)
- Hsu, C., Lu, H.: Why do people play on-line games? An extended TAM with social influences and flow experience. *Inf. Manag.* **41**, 853–868 (2004)
- Hu, L.-T., Bentler, P.: Cutoff criteria for fit indexes in covariance structure analysis: conventional criteria versus new alternatives. *Struct. Equ. Model: A Multidiscip. J.* **6**, 1–55 (1999)
- Hu, R., Pu, P.: A comparative user study on rating vs. personality quiz based preference elicitation methods. In: 14th International Conference on Intelligent User Interfaces, pp. 367–371. Sanibel Island, FL (2009)
- Hu, R., Pu, P.: A study on user perception of personality-based recommender systems. In: 18th International Conference on User Modeling, Adaptation, and Personalization, LNCS, vol. 6075. pp. 291–302. Big Island, HI (2010)
- Hu, R., Pu, P.: Enhancing recommendation diversity with organization interfaces. In: 16th International Conference on Intelligent User Interfaces, pp. 347–350. Palo Alto, CA (2011)
- Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. In: 2008 Eighth IEEE International Conference on Data Mining, pp. 263–272. Pisa, Italy (2008)
- Huang, P., Lurie, N.H., Mitra, S.: Searching for experience on the web: an empirical examination of consumer behavior for search and experience goods. *J. Market.* **73**(2), 55–69 (2009)
- Iyengar, S., Lepper, M.: When choice is demotivating: can one desire too much of a good thing? *J. Pers. Soci. Psychol.* **79**, 995–1006 (2000)
- Jones, N., Pu, P., Chen, L.: How users perceive and appraise personalized recommendations. In: 17th International Conference on User Modeling, Adaptation, and Personalization Conference, vol. 5535, pp. 461–466. Trento, Italy, LNCS (2009)
- Kamis, A., Davern, M.J.: Personalizing to product category knowledge: exploring the mediating effect of shopping tools on decision confidence. In: 37th Annual Hawaii International Conference on System Sciences. Big Island, HI (2004)
- Kaplan, B., Duchon, D.: Combining qualitative and quantitative methods in information systems research: a case study. *Mis Q.* **12**, 571–586 (1988)
- Kautz, H., Selman, B., Shah, M.: Referral Web: combining social networks and collaborative filtering. *Commun. ACM* **40**, 63–65 (1997)
- Knijnenburg, B.P., Reijmer, N.J.M., Willemsen, M.C.: Each to his own: how different users call for different interaction methods in recommender systems. In: 5th ACM Conference on Recommender Systems, pp. 141–148. Chicago, IL (2011)
- Knijnenburg, B.P., Willemsen, M.C.: Understanding the effect of adaptive preference elicitation methods on user satisfaction of a recommender system. In: Third ACM Conference on Recommender Systems, pp. 381–384. New York, NY (2009)
- Knijnenburg, B.P., Willemsen, M.C.: The effect of preference elicitation methods on the user experience of a recommender system. In: 28th International Conference on Human Factors in Computing Systems, pp. 3457–3462. Atlanta, GA (2010)

- Knijnenburg, B.P., Willemsen, M.C., Kobsa, A.: A pragmatic procedure to support the user-centric evaluation of recommender systems. In: 5th ACM Conference on Recommender Systems, pp. 321–324. Chicago, IL (2011a)
- Knijnenburg, B.P., Meesters, L., Marrow, P., Bouwhuis, D.: User-centric evaluation framework for multimedia recommender systems. In: First International Conference on User Centric Media, LNCS, vol. 40, pp. 366–369. Venice, Spain (2010a)
- Knijnenburg, B.P., Willemsen, M.C., Hirtbach, S.: Receiving recommendations and providing feedback: The user-experience of a recommender system. In: 11th International Conference on Electronic Commerce and Web Technologies, LNBP, vol. 61, pp. 207–216. Bilbao, Spain (2010b)
- Kobsa, A., Teltzrow, M.: Contextualized communication of privacy practices and personalization benefits: impacts on users' data sharing and purchase behavior. In: Workshop on Privacy Enhancing Technologies, LNCS, vol. 3424, pp. 329–343. Toronto, Canada (2005)
- Komiak, S.Y.X., Benbasat, I.: The effects of personalization and familiarity on trust and adoption of recommendation agents. *Mis Q.* **30**, 941–960 (2006)
- Konstan J.A., Riedl J.: Recommender Systems: From Algorithms to User Experience. *User Model. User-Adap. Inter.* **22**(1–2), 101–123 (2012)
- Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. *IEEE Comput.* **42**, 30–37 (2009)
- Koren, Y.: Factor in the neighbors: Scalable and accurate collaborative filtering. *Trans. Knowl. Discov. Data* **4**, 1–24 (2010)
- Koufaris, M.: Applying the technology acceptance model and flow theory to online consumer behavior. *Inf. Syst. Res.* **13**, 205–223 (2003)
- Kramer, T.: The effect of measurement task transparency on preference construction and evaluations of personalized recommendations. *J. Market. Res.* **44**, 224–233 (2007)
- Krishnan, V., Narayanan Shetty, P., Nathan, M., Davies, R., Konstan, J.: Who predicts better?: Results from an online study comparing humans and an online recommender system. In: 2008 ACM Conference on Recommender Systems, pp. 211–218. Lausanne, Switzerland (2008)
- Law, E., Roto, V., Hassenzahl, M., Vermeeren, A., Kort, J.: Understanding, scoping and defining user experience: a survey approach. In: 27th International Conference on Human Factors in Computing Systems, pp. 719–728. Boston, MA (2009)
- Lynch, J.G. Jr., Ariely, D.: Wine online: search cost and competition on price, quality, and distribution. *Market. Sci.* **19**, 83–103 (2000)
- Marrow, P., Hanbridge, R., Rendle, S., Wartena, C., Freudenthaler, C.: MyMedia: producing an extensible framework for recommendation. In: Networked Electronic Media Summit 2009. Saint-Malo, France (2009)
- McNamara, N., Kirakowski, J.: Functionality, usability, and user experience: three areas of concern. *ACM Interact.* **13**, 26–28 (2006)
- McNee, S., Albert, I., Cosley, D., Gopalkrishnan, P., Lam, S., Rashid, A., Konstan, J., Riedl, J.: On the recommending of citations for research papers. In: 2002 ACM Conference on Computer Supported Cooperative Work, pp. 116–125. New Orleans, LA (2002)
- McNee, S., Riedl, J., Konstan, J.: Being accurate is not enough: how accuracy metrics have hurt recommender systems. In: 24th International Conference Human factors in computing systems, pp. 1097–1101. Montréal, Canada (2006a)
- McNee, S., Riedl, J., Konstan, J.: Making recommendations better: an analytic model for human-recommender interaction. In: 24th International Conference Human factors in computing systems, pp. 1103–1108. Montréal, Canada (2006b)
- Meesters, L., Marrow, P., Knijnenburg, B.P., Bouwhuis, D., Glancy, M.: MyMedia Deliverable 1.5 End-user Recommendation Evaluation Metrics (2008) [http://www.mymediaproject.org/Publications/WP1/MyMedia\\_D1.5.pdf](http://www.mymediaproject.org/Publications/WP1/MyMedia_D1.5.pdf)
- Murray, K., Häubl, G.: Interactive consumer decision aids. In: Wierenga, B. (ed.) *Handbook of Marketing Decision Models*, pp. 55–77. Springer, Heidelberg (2008)
- Murray, K., Häubl, G.: Personalization without interrogation: towards more effective interactions between consumers and feature-based recommendation agents. *J. Interact. Market.* **23**, 138–146 (2009)
- Muthén, B.: A general structural equation model with dichotomous, ordered categorical, and continuous latent variable indicators. *Psychometrika* **49**, 115–132 (1984)
- Nelson, P.: Information and consumer behavior. *J. Polit. Econ.* **78**, 311–329 (1970)

- Netemeyer, R., Bentler, P.: Structural equations modeling and statements regarding causality. *J. Consum. Psychol.* **10**, 83–85 (2001)
- Ochi, P., Rao, S., Takayama, L., Nass, C.: Predictors of user perceptions of web recommender systems: how the basis for generating experience and search product recommendations affects user responses. *Int. J. Hum.-Comput. Stud.* **68**, 472–482 (2010)
- Ozok, A.A., Fan, Q., Norcio, A.F.: Design guidelines for effective recommender system interfaces based on a usability criteria conceptual model: results from a college student population. *Behav. Inf. Technol.* **29**, 57–83 (2010)
- Paramythios, A., Weibelzahl, S., Masthoff, J.: Layered evaluation of interactive adaptive systems: framework and formative methods. *User Model. User-Adap. Inter.* **20**, 383–453 (2010)
- Pathak, B., Garfinkel, R., Gopal, R.D., Venkatesan, R., Yin, F.: Empirical analysis of the impact of recommender systems on sales. *J. Manag. Inf. Syst.* **27**, 159–188 (2010)
- Pedersen, P.: Behavioral effects of using software agents for product and merchant brokering: an experimental study of consumer decision-making. *Int. J. Electron. Commer.* **5**, 125–141 (2000)
- Pommeranz, A., Broekens, J., Wiggers, P., Brinkman, W.-P., Jonker, C. M.: Designing interfaces for explicit preference elicitation: a user-centered investigation of preference representation and elicitation process. *User Model. User-Adap. Inter.* **22** (2012). doi:[10.1007/s11257-011-9116-6](https://doi.org/10.1007/s11257-011-9116-6)
- Preece, J., Rogers, Y., Sharp, H.: *Interaction Design: Beyond Human-Computer Interaction*. Wiley, New York (2002)
- Pu, P., Chen, L.: Trust building with explanation interfaces. In: 11th International Conference on Intelligent User Interfaces, pp. 93–100. Sydney, Australia (2006)
- Pu, P., Chen, L.: Trust-inspiring explanation interfaces for recommender systems. *Knowl.-Based Syst.* **20**, 542–556 (2007)
- Pu, P., Chen, L.: A user-centric evaluation framework of recommender systems. In: ACM RecSys 2010 Workshop on User-Centric Evaluation of Recommender Systems and Their Interfaces, pp. 14–21. Barcelona, Spain (2010)
- Pu, P., Chen, L., Hu, R.: Evaluating recommender systems from the user's perspective: survey of the state of the art. *User Model. User-Adap. Inter.* **22** (2012). doi:[10.1007/s11257-011-9115-7](https://doi.org/10.1007/s11257-011-9115-7)
- Pu, P., Chen, L., Kumar, P.: Evaluating product search and recommender systems for E-commerce environments. *Electron. Commer. Res.* **8**, 1–27 (2008)
- Rendle, S., Schmidt-Thieme, L.: Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In: 2008 ACM Conference on Recommender systems, pp. 251–258. Lausanne, Switzerland (2008)
- Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L.: BPR: Bayesian personalized ranking from implicit feedback. In: Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, pp. 452–461. Montreal, Canada (2009)
- Resnick, P., Varian, H.: Recommender systems. *Commun. ACM* **40**, 56–58 (1997)
- Scheibehenne, B., Greifeneder, R., Todd, P.: Can there ever be too many options? A meta-analytic review of choice overload. *J. Consum. Res.* **37**, 409–425 (2010)
- Schwartz, B.: *The Paradox of Choice: Why More Is Less*. HarperCollins, New York (2004)
- Senecal, S., Nantel, J.: The influence of online product recommendations on consumers' online choices. *J. Retail.* **80**, 159–169 (2004)
- Sheeran, P.: Intention-behavior relations: a conceptual and empirical review. *Eur. Rev. Soc Psychol.* **12**, 1–36 (2002)
- Simonson, I., Tversky, A.: Choice in context: tradeoff contrast and extremeness aversion. *J. Market. Res.* **29**, 281–295 (1992)
- Spiekermann, S., Grossklags, J., Berendt, B.: E-privacy in 2nd generation E-commerce: privacy preferences versus actual behavior. In: 3rd ACM Conference on Electronic Commerce, pp. 38–47. Tampa, FL (2001)
- Stolze, M., Nart, F.: Well-integrated needs-oriented recommender components regarded as helpful. In: 22nd International Conference on Human Factors in Computing Systems, p. 1571. Vienna, Austria (2004)
- Tam, K.Y., Ho, S.Y.: Web personalization as a Persuasion strategy: an elaboration likelihood model perspective. *Inf. Syst. Res.* **16**, 271–291 (2005)
- Teltzrow, M., Kobsa, A.: Impacts of user privacy preferences on personalized systems. *Hum.-Comput. Interact. Ser.* **5**, 315–332 (2004)

- Tintarev, N., Masthoff, J.: Evaluating the Effectiveness of Explanations for Recommender Systems. *User Model. User-Adap. Inter.* **22** (2012). doi:[10.1007/s11257-011-9117-5](https://doi.org/10.1007/s11257-011-9117-5)
- Torres, R., McNee, S., Abel, M., Konstan, J., Riedl, J.: Enhancing digital libraries with TechLens+. In: 4th ACM/IEEE-CS Joint Conference on Digital Libraries, pp. 228–236. Tucson, AZ (2004)
- Van Velsen, L., VanDer Geest, T., Klaassen, R., Steehouder, M.: User-centered evaluation of adaptive and adaptable systems: a literature review. *Knowl. Eng. Rev.* **23**, 261–281 (2008)
- Venkatesh, V., Morris, M., Davis, G., Davis, F.: User acceptance of information technology: toward a unified view. *Mis Q.* **27**, 425–478 (2003)
- Viappiani, P., Faltings, B., Pu, P.: Preference-based search using example-critiquing with suggestions. *J. Artif. Intell. Res.* **27**, 465–503 (2006)
- Viappiani, P., Pu, P., Faltings, B.: Preference-based search with adaptive recommendations. *AI Commun.* **21**, 155–175 (2008)
- Vijayasarathy, L.R., Jones, J.M.: Do internet shopping aids make a difference? an empirical investigation. *Electron. Markets* **11**, 75–83 (2001)
- Wang, W., Benbasat, I.: Recommendation agents for electronic commerce: effects of explanation facilities on trusting beliefs. *J. Manag. Inf. Syst.* **23**, 217–246 (2007)
- Willemsen, M.C., Knijnenburg, B.P., Graus, M.P., Velter-Bremmers, L.C.M., Fu, K.: Using latent features diversification to reduce choice difficulty in recommendation lists. In: RecSys'11 Workshop on Human Decision Making in Recommender Systems, CEUR-WS, vol. 811, pp. 14–20. Chicago, IL (2011)
- Xiao, B., Benbasat, I.: ECommerce product recommendation agents: use, characteristics, and impact. *Mis Q.* **31**, 137–209 (2007)
- Yu, J., Ha, I., Choi, M., Rho, J.: Extending the TAM for a t-commerce. *Inf. Manag.* **42**, 965–976 (2005)
- Ziegler, C., McNee, S., Konstan, J., Lausen, G.: Improving recommendation lists through topic diversification. In: 14th international World Wide Web Conference, pp. 22–32. Chiba, Japan (2005)
- Zins, A., Bauernfeind, U.: Explaining online purchase planning experiences with recommender websites. In: International Conference on Information and Communication Technologies in Tourism, pp. 137–148. Innsbruck, Austria (2005)

## Author Biographies

**Bart P. Knijnenburg** is a Ph.D. candidate in Informatics at the University of California, Irvine. His work focuses on the user experience of recommender systems, adaptive preference elicitation methods, and privacy-aware interfaces for adaptive systems. He received his B.S. degree in Innovation Sciences and his M.S. degree in Human-Technology Interaction from Eindhoven University of Technology (TU/e), The Netherlands, and his M.A. degree in Human-Computer Interaction from Carnegie Mellon University. The work described in the current paper was conducted while employed as a researcher on the MyMedia project at TU/e.

**Martijn C. Willemsen** holds an M.S. in Electrical engineering and in Technology and Society, and a Ph.D. degree in cognitive psychology (decision making) from the TU/e. Since 2002 he is an assistant professor in the HTI group at TU/e. From 2003-2004 he was a visiting PostDoc at Columbia University (New York) to work with Prof. Eric Johnson on online (web-based) process tracing methods. His expertise is in online consumer behavior and decision-making, and in user-centric evaluation. His current research includes the application of the psychology of decision making in recommender systems, and on this topic he served as an advisor in the MyMedia project from which the current paper originates.

**Zeno Gantner** is a researcher at University of Hildesheim's Information Systems and Machine Learning Lab. He received a Dipl.-Inf. degree (Master in computer science) from University of Freiburg, Germany. His research focus is machine-learning techniques for recommender systems. He is the main author of the MyMediaLite recommender algorithm library.

**Hakan Soncu** is a Software Development Engineer at EMIC. He received his M.S. in Software Systems Engineering from RWTH Aachen.

**Chris Newell** is a Lead Technologist at BBC Research and Development. He received his B.A. in Physics and D.Phil. in Engineering from the University of Oxford. His primary interests are user interfaces and the associated metadata in digital broadcasting systems and he has contributed to the DVB, ID3 and TV-Anytime standards.



[AT&T Research Home](#)

[Statistics Research](#)

[Information Visualization Research](#)

[Employment at AT&T Research](#)

**August, 2009.** Please see [here](#) for details about our exciting run to the Netflix Grand Prize - combining with Pragmatic Theory and Big Chaos to win the \$1M Grand Prize!

**Previous updates about our team:**

**December 10, 2008** We are proud to be the recipient's of Netflix' [2008 Progress Prize!](#). This year, we combined with our colleagues at [Commendo Research](#) in Austria to get to a 9.44% improvement over Netflix' Cinematch algorithm.

**November, 2008** Our BellKor Team was featured in a recent issue of the New York Times Magazine! [Read the article online](#) or check out the accompanying [video](#).

**March 21, 2008:** [An update](#) on our progress in the Netflix Prize.

#### Team BellKor/KorBell - Netflix Progress Prize Winners



Team BellKor is made up of [Bob Bell](#) and [Chris Volinsky](#), from the [Statistics Research](#) group in AT&T Labs, and [Yehuda Koren](#), who recently left AT&T Labs for Yahoo! Research in Israel. Our team has won the first two \$50,000 Progress Prizes awarded by Netflix as part of their \$1 Million competition to improve their recommendation algorithm. Check out their current performance on the Netflix [leaderboard](#), and watch this AT&T Tech Channel [interview](#) with researchers Bob Bell and Chris Volinsky.

With the [best score](#) at the one year anniversary of the competition, they [won](#) the coveted Progress Prize. Netflix sent a cool commemorative [plaque](#), although at a square meter in size, and weighing almost 200 pounds it is more like a monument!



**Papers describing our techniques in analyzing the data and our experiences in the competition:**

- [The BellKor 2008 Solution to the Netflix Prize](#). This is the document which lays out our overall strategy - as was required in the [rules](#) of the competition in order to claim the Progress Prize.
- [Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model](#). **KDD 2008..**
- [Recent Progress in Collaborative Filtering](#). **RecSys 2008**
- [Factor in the Neighbors: Scalable and Accurate Collaborative Filtering](#). **submitted**
- [Chasing \\$1,000,000: How We Won The Netflix Progress Prize](#). **ASA Statistical and Computing Graphics Newsletter**. Volume 18, Number 2.
- [Lessons from the Netflix Prize Challenge](#). **SIGKDD Explorations**, Volume 9, Issue 2.
- [The BellKor Solution to the Netflix Prize](#). This is the document which lays out our overall strategy - as was required in the [rules](#) of the competition in order to claim the Progress Prize.

- Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. **ICDM 2007**.
- Improved Neighborhood Based Collaborative Filtering. **KDD 2007 Netflix Competition Workshop**.
- Modelling relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems. **KDD 2007**.

See some more detail about our Netflix entry [here](#)

Find press coverage [here](#)

---

### Thank You

We would like to thank Netflix for putting on a heck of a competition. The nature of the competition and especially the collaborative spirit of the participants has made this an exciting and rewarding scientific endeavor. We also would like to thank all of our competitors on the leaderboard for pushing us harder and forcing us to come up with the best possible solution. We look forward to chasing the million bucks with you all!

Please email us at [bellkor@research.att.com](mailto:bellkor@research.att.com)

For site issues, contact [research-info@research.att.com](mailto:research-info@research.att.com).

Legal Notice

© 2011 AT&T Intellectual Property. All Rights Reserved. AT&T and the AT&T logo are trademarks of AT&T Intellectual Property.

# The BellKor 2008 Solution to the Netflix Prize

Robert M. Bell  
AT&T Labs - Research  
Florham Park, NJ

Yehuda Koren  
Yahoo! Research  
Haifa, Israel

Chris Volinsky  
AT&T Labs - Research  
Florham Park, NJ

[BellKor@research.att.com](mailto:BellKor@research.att.com)

## 1. Introduction

Our RMSE=0.8643<sup>2</sup> solution is a linear blend of over 100 results. Some of them are new to this year, whereas many others belong to the set that was reported a year ago in our 2007 Progress Prize report [3]. This report is structured accordingly. In Section 2 we detail methods new to this year. In general, our view is that those newer methods deliver a superior performance compared to the methods we used a year ago. Throughout the description of the methods, we highlight the specific predictors that participated in the final blended solution. Nonetheless, the older methods still play a role in the blend, and thus in Section 3 we list those methods repeated from a year ago. Finally, we conclude with general thoughts in Section 4.

## 2. New Methods

The foundations of our progress during 2008 are laid out in the KDD 2008 paper [4]. The significant enhancement of the techniques reported in that paper is accounting for temporal effects in the data. In the following we briefly review the techniques described in the paper [4], while giving extra details on how those methods can address temporal effects, and some other variants that we tried. For a deeper treatment and general background, please refer to the original paper. We assume a good familiarity with our notation at [4] and with last year's Progress Prize Report [3]. All methods described in this section are trained using standard stochastic gradient descent, which became a preferred framework for analyzing the Netflix dataset. This algorithm requires setting two constants – step size (aka, learning rate) and regularization coefficient (aka, weight decay). We derived the values of these constants manually, seeking to minimize RMSE on the Probe set. A description of the learning equations and proper constant settings are given in the original papers [4,5].

### 2.1 Factor models

In the paper [4] we give a detailed description of three factor models. The first one is a simple SVD with biases model as in Eq. (12) of the paper:

---

<sup>2</sup> All root mean squared error (RMSE) mentioned in this article are measured on the Netflix Quiz set.

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T q_i$$

This model is now widely used among Netflix competitors, as evident by Netflix Prize Forum posts, and is formally described by others [6, 7]. Hereinafter, we will refer to this model as “SVD”, in accordance with the terminology at [4].

The second model delivers a similar accuracy, while offering several practical advantages, as described in Eq. (13) of the paper:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left( |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} (r_{uj} - b_{uj}) x_j + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right)$$

As in [4], we will refer to this model as “Asymmetric-SVD”. Interestingly, it can be shown that this is a factorized neighborhood model in disguise [5]. Thus, this model bridges neighborhood and factor models.

Finally, the more accurate factor model, to be named “SVD++”, is as described in Eq. (15) of [4]:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left( p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right)$$

These models are learnt using stochastic gradient descent. The variable  $\mu$  is constant (mean rating in training data, e.g.,  $\mu = 3.7$ ). However, the user- and movie-biases  $b_u, b_i$ , are usually learnt from the data to improve prediction accuracy.

A single solution in our blend is based on the SVD++ model with 60 factors. In this case, user- and movie-biases were fixed as constants, which reduces prediction accuracy and is equivalent to running SVD++ on residuals of double-centered data. This leads to RMSE=0.8966.

## Accounting for temporal effects

We identify three strong temporal effects in the data:

1. Movie biases – movies go in and out of popularity over time. Several events can cause a movie to become more or less favorable. This is manifested in our models by the fact that movie bias  $b_i$  is not a scalar but a function that changes over time. This effect is relatively easy to capture, because such changes span extended amounts of time. That is, we do not expect a movie likeability to hop on a daily basis, but rather to change over more extended periods. Further, we have

- relatively many ratings per movie, what allows us to model these effects adequately.
2. User biases – users change their baseline ratings over time. For example, a user who tended to rate an average movie “4 stars”, may now rate such a movie “3 stars”. This means that in our models we would like to take the parameter  $b_u$  as a function that changes over time. Such effects can stem from many reasons. For example, it is related to a natural drift in a user’s rating scale, to the fact that ratings are given in relevance to other ratings that were given recently and also to the fact that the identity of the rater within a household can change over time. Importantly, this effect is characterized with two properties that make it hard to be captured. First, we observe the effect even at the resolution of a single day, which is the finest resolution available within the Netflix data. In other words, the effective user bias on a day can be significantly different than the user bias on the day earlier or the day after. Second difficulty stems from the fact that users are usually associated with only a handful of ratings, especially when focusing on their ratings within a single day.
  3. User preferences – users change their preferences over time. For example, a fan of the “psychological thrillers” genre may become a fan of “crime dramas” a year later. Similarly, humans change their perception on certain actors and directors. Part of this effect is also related to the fact that several people may rate within the same household. This effect is modeled by taking the user factors (the vector  $p_u$ ) as a function that changes over time. Once again, we need to model those changes at the very fine level of a daily basis (after all, at each new session we may receive the ratings from a different person at the household), while facing the built-in scarcity of user ratings. In fact, these temporal effects are the hardest to capture, because preferences are not as pronounced as main effects (user-bias), but are split over many factors.

Now, let us describe how those temporal effects were inserted into our models. We focus on the more accurate model, which is “SVD++”. The general framework is:

$$\hat{r}_{ui}(t) = \mu + b_u(t) + b_i(t) + q_i^T \left( p_u(t) + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j \right)$$

Here, we predict a rating at time (or, day)  $t$ . Notice that the relevant parameters are now structured as time-dependent functions, which are defined as described shortly. To increase accuracy, it is important that all parameters will be learnt from the data simultaneously. In other words, biases, movie-factors and user-factors are jointly learned from the data.

As mentioned earlier, temporal effects of movie biases are easier to catch since we do not need the finest resolution there, and since there are many ratings associated with a single movie. Thus, an adequate decision would be to split the movie biases into time-based bins. We are using 30 bins, spanning all days in the dataset: from Dec 31, 1999 till Dec 31, 2005, such that each bin corresponds to about 10 consecutive weeks of data. This effectively increases the number of parameters required for describing movies biases by a

factor of 30. Each day,  $t$ , is associated with an integer between 1 to 30 called  $\text{Bin}(t)$ , such that:

$$b_i(t) = b_{i,\text{Bin}(t)}$$

While binning the parameters works well on the movies, it is more of a challenge on the user side. On one hand, we would like a finer resolution for users to detect very short lived temporal effects. On the other hand, we do not expect having enough ratings per user to produce reliable estimates for isolated bins. Different function forms can be considered for modeling temporal user behavior. Their prediction accuracy is related to the number of involved parameters. We concentrate on two simple extreme choices, as we have found that their sum, gave us almost as good results as we could get by other options that we tried. We start dealing with user biases ( $b_u$ 's); user preferences ( $p_u$ 's) will be treated analogously.

The first modeling choice is very concise in number of parameters, and requires adding only a single parameter per user bias. Let us first introduce some new notation. For each user  $u$ , let us denote the mean date of rating by  $t_u$ . Now, if  $u$  rated a movie on day  $t$ , then the associated time deviation of this rating is defined as:

$$\text{dev}_u(t) = \text{sign}(t - t_u) \cdot |t - t_u|^\beta$$

We set the value of  $\beta$  by cross validation to 0.4. Then, for each user we center all those time deviations, and work with the centered variables  $\widehat{\text{dev}}_u(t)$ . Notice that those variables are constants that are derived directly from the training data.

Now, in order to define a time dependent bias, we introduce a single new parameter for each user called  $\alpha_u$  and get our first definition of a time-dependent user-bias:

$$b_u^{(1)}(t) = b_u + \alpha_u \cdot \widehat{\text{dev}}_u(t)$$

This offers a simple linear model that does not require adding many new parameters, but at the same time is quite limited in its flexibility. Therefore, we also resort to another extreme, the most flexible model, where we assign a single parameter per user and day such that the user biases become:

$$b_u^{(2)}(t) = b_{u,t}$$

This way, on each day a user bias is captured by an independent parameter. In the Netflix data, a user rates on 40 different days on average. Thus, working with  $b_u^{(2)}(t)$  requires, on average, 40 parameters to describe each user bias (unlike  $b_u^{(1)}(t)$  that required two parameters per user bias). In fact,  $b_u^{(2)}(t)$  is inadequate as a standalone for capturing the user bias, since it misses all sorts of signal that span more than a single day. Thus, in practice we add it to the other kind of time-dependent user bias, obtaining:

$$b_u^{(3)}(t) = b_u^{(1)}(t) + b_u^{(2)}(t)$$

The same way we treat user biases we can also treat each component of the user preferences  $p_u(t)^T = (p_{u1}(t), p_{u2}(t), \dots, p_{uf}(t))$ . Either as:

$$p_{uk}^{(1)}(t) = p_{uk} + \alpha_{uk} \cdot \widehat{dev}_u(t) \quad k = 1, \dots, f$$

Or:

$$p_{uk}^{(3)}(t) = p_{uk}^{(1)}(t) + p_{uk,t} \quad k = 1, \dots, f$$

Notice that for the Netflix data, taking the user factors as  $p_{uk}^{(3)}(t)$  requires, on average, about 42 parameters per component. This can lead to tremendous space requirements, and would render this particular variant less attractive under many real life situations. In fact, we came up with more concise models of almost the same accuracy. However, for the sake of the Netflix competition, the most elaborate description of user factors was found useful. It is interesting to comment that a simple regularized stochastic gradient descent algorithm was enough to avoid overfitting, and quite incredibly allowed us to fit many billions of parameters to the data.

We implemented the most elaborated time-dependent SVD++ model, which we henceforth dub “SVD++<sup>(3)</sup>”:

$$\hat{r}_{ui}(t) = \mu + b_u^{(3)}(t) + b_i(t) + q_i^T \left( p_u^{(3)}(t) + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j \right)$$

As stated earlier, all involved parameters (biases, user- and movie-factors) are learnt simultaneously, such that the model is trained directly on the raw data, without any kind of pre-processing; see [4]. Prediction accuracy slowly improves, with increasing number of factors, as shown in the following table:

| $f$ – dimensionality of factor vectors | RMSE   |
|--|--------|
| 20                                     | 0.8893 |
| 50                                     | 0.8831 |
| 100                                    | 0.8812 |
| 200                                    | 0.8806 |
| 500                                    | 0.8801 |
| 1000                                   | 0.8798 |
| 2000                                   | 0.8795 |

**Table 1.** Accuracy of time dependent SVD++ model

The solution included in the blend is based on 2000 factors yielding RMSE=0.8795. We also implemented “lighter” variants of time-dependent SVD++, which required far less parameters, such as:

$$\hat{r}_{ui}(t) = \mu + b_u^{(1)}(t) + b_i(t) + q_i^T \left( p_u^{(1)}(t) + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j \right)$$

Henceforth, we will name this model “SVD++<sup>(1)</sup>”. We applied it with 100 factors and got RMSE=0.8879 (not included in the final blend). Then, we took the residuals

of this model, and smoothed them by a movie-movie neighborhood model, as described in our ICDM'2007 paper [1] (or, KDD-Cup'2007 paper [2]). This neighborhood model is denoted as [kNN] in Sec. 3 (or in [3]). We used 30 neighbors, and the RMSE of the result was 0.8842 (included in the final blend). We also included in the blend another related variant, where the inner product matrix was estimated over the residuals of an RBM, what lowered the RMSE to 0.8822.

## 2.2 Neighborhood models

We implemented various variants of the neighborhood model described in Sec. 3 of [4]. The basic model is based on Eq. (9) there:

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathcal{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}(u)} (r_{uj} - b_{uj}) w_{ij} + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} c_{ij}$$

The result of RMSE=0.9002 was included in the final blend.

Other variants of the model would use only a subset of the neighbors, and can be applied on residuals of other methods. The general formula based on Eq. (10) in [4] is:

$$\hat{r}_{ui} = \tilde{b}_{ui} + |\mathcal{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathcal{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}^k(i;u)} c_{ij}$$

Here,  $\tilde{b}_{ui}$  is the prediction for the rating by user  $u$  and movie  $i$ , as estimated by some other method, and  $k$  is the number of neighbors. We will refer to this model as “GlobalNgbr”.

This way we applied the kNN model (with full set of neighbors;  $k=17,770$ ), to residuals of SVD++ with 60 factors (and fixed biases). The result, with RMSE=0.8906 is included in the final blend.

We also applied this method with  $k=2000$  on residuals of global effects to obtain RMSE=0.9067 (also, within the blend).

The previous prediction rule can be easily extended to address time-dependent user biases:

$$\hat{r}_{ui}(t) = \tilde{b}_{ui} + b_u^{(3)}(t) + |\mathcal{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathcal{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}^k(i;u)} c_{ij}$$

Within the final blend, such a model (with  $k=35$ ) was applied to residuals of a Restricted Boltzmann Machine (100 hidden units) to obtain an RMSE of 0.8893.

Similarly, the basic model (which works on raw ratings) can be enhanced to account for time dependent user- and movie-biases:

$$\hat{r}_{ui}(t) = \mu + b_u^{(3)}(t) + b_i(t) + |\mathcal{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}(u)} (r_{uj} - b_{uj}) w_{ij} + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} c_{ij}$$

Another slight improvement is obtained by decaying neighbors that were rated distantly in time, by adding another term to the prediction rule:

$$\begin{aligned} \hat{r}_{ui}(t) &= \mu + b_u^{(3)}(t) + b_i(t) + |\mathcal{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}(u)} (r_{uj} - b_{uj}) w_{ij} + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} c_{ij} \\ &\quad + \sum_{j \in \mathcal{R}(u)} \exp(\gamma \cdot |t - t_{uj}|) d_{ij} \end{aligned}$$

Here,  $t_{uj}$  is the day in which user  $u$  rated item  $j$ , and  $|t - t_{uj}|$  is the number of days between the rating of item  $i$  and that of item  $j$ . The constant  $\gamma$  was set to 0.5.

The result of this neighborhood model, as included in the final blend, is of RMSE=0.8914.

An alternative version of the neighborhood model used the Sigmoid function in order to aggregate the movie-movie weights, as follows:

$$\hat{r}_{ui} = \tilde{b}_{ui} + \lambda \cdot \sigma \left( \sum_{j \in \mathcal{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + \sum_{j \in \mathcal{N}^k(i;u)} c_{ij} \right)$$

The function  $\sigma(x) = (1 + \exp(-x))^{-1} - 0.5$  maps  $x$  to (-0.5,0.5). The parameter  $\lambda$  is learnt from the data together with all other parameters.

Results of this method are inferior to those of the methods described in [4] and earlier in this subsection. Nonetheless, three related results are used within the final blend. First, we applied the method to residuals of global effects with  $k=17770$  (full dense set of neighbors) to obtain RMSE=0.9200. When limiting the number of neighbors using  $k=200$ , the resulting RMSE increases to 0.9230. The last variant was applied to residuals of RBM (200 hidden units) to yield RMSE=0.8931.

## 2.3 Integrated models

As we explain in [4] (Sec. 5), one can achieve better prediction accuracy by combining the neighborhood and factor models. In particular, the neighborhood model described in the previous subsection allows a symmetric treatment, where neighborhood parameters and factor parameters are learnt simultaneously. The basic model follows Eq. (16) of [4]:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left( p_u + |\mathbf{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{N}(u)} y_j \right) \\ + |\mathbf{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathbf{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{N}^k(i;u)} c_{ij}$$

We will later refer to this model as “Integrated”. It can be enhanced to account for temporal effects, as we did with the factor models. We start with the more concise models, which require a modest addition of parameters to achieve:

$$\hat{r}_{ui}(t) = \mu + b_u^{(1)}(t) + b_i(t) + q_i^T \left( p_u^{(1)}(t) + |\mathbf{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{N}(u)} y_j \right) \\ + |\mathbf{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathbf{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{N}^k(i;u)} c_{ij}$$

The result of this model with  $f=750$  and  $k=300$ , as included in the final blend, yields RMSE=0. 8827.

In order to further improve accuracy, we employ a more elaborated temporal model for the user biases:

$$\hat{r}_{ui}(t) = \mu + b_u^{(3)}(t) + b_i(t) + q_i^T \left( p_u^{(1)}(t) + |\mathbf{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{N}(u)} y_j \right) \\ + |\mathbf{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathbf{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{N}^k(i;u)} c_{ij}$$

Once again, we use a limited neighborhood size ( $k=300$ ), as neighborhood models better complement factor models when they are well localized. Prediction accuracy very slowly improves when increasing the dimensionality of the factor model, as shown in the following table:

| $f$  | RMSE   |
|------|--------|
| 200  | 0.8789 |
| 500  | 0.8787 |
| 750  | 0.8786 |
| 1000 | 0.8785 |
| 1500 | 0.8784 |

**Table 2.** Accuracy of an integrated model

The result with  $f=1500$  (RMSE=0.8784) is included in the final blend.

We also tried to integrate the neighborhood model with other factor models. Two related results are in the final blend. First, we added the neighborhood model ( $k=300$ ) to an Asymmetric-SVD model ( $f=60$ ), with no temporal effects. The achieved RMSE was 0.8959. Second, we added the neighborhood model ( $k=300$ ) to an RBM with Gaussian visible units and 256 hidden units. The resulting RMSE was 0.8943 (once again, temporal effects were not addressed here).

We should note that we have not tried (yet) the supposedly most powerful integrated model, which addresses full temporal effects also for user preferences, by replacing  $p_u^{(1)}(t)$  with  $p_u^{(3)}(t)$ , as follows:

$$\begin{aligned}\hat{r}_{ui}(t) = & \mu + b_u^{(3)}(t) + b_i(t) + q_i^T \left( p_u^{(3)}(t) + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j \right) \\ & + |\mathcal{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathcal{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}^k(i;u)} c_{ij}\end{aligned}$$

## 2.3 Other methods

There are two additional developments during the last year, with a very modest contribution to the final blend.

### 2.3.1 Shrinking towards recent actions

A possible way for accounting for temporal effects is by overweighting the more recent user actions. Indeed, this is inferior compared to the more principled approach described earlier, which could provide a full modeling of how user behavior is changing over time. Nonetheless, when holding prediction sets that have been previously computed without accounting for temporal effects, a simple correction as described below is effective.

In the following we assume that we want to correct  $\hat{r}_{ui}$ , which is the predicted rating for user  $u$  on item  $i$  at day  $t (= t_{ui})$ . We would like to shrink  $\hat{r}_{ui}$  towards the average rating of  $u$  on day  $t$ . The rationale here is that the single day effect is among the strongest temporal effects in the data. To this end we compute several magnitudes related to the actions of user  $u$  on day  $t$ :

- $n_{ut}$  - the number of ratings  $u$  gave on day  $t$
- $\bar{r}_{ut}$  - the mean rating of  $u$  at day  $t$
- $V_{ut}$  - the variance of  $u$ 's ratings at day  $t$

This allows us to compute a confidence coefficient, related to how  $u$  tended to concentrate his/her rating on day  $t$ :

- $c_{ut} = n_{ut} \cdot \exp(-\alpha \cdot V_{ut})$

Accordingly, we shrink our estimate towards  $\bar{r}_{ut}$  controlled by the confidence coefficient, so that the corrected prediction is:

$$\frac{\beta \cdot \hat{r}_{ui} + c_{ut} \cdot \bar{r}_{ut}}{\beta + c_{ut}}$$

The participating constants were determined by cross validation to be:  $\alpha = 8, \beta = 11$ .

We used this correction with a single solution in the final blend. First, we combined two solutions. The first one is 50 neighbors kNN on 100-unit RBM with RMSE 0.8888 (see predictor #40 in last year's Progress Prize Report [3]). Second result is by the SVD++<sup>(1)</sup> model with  $f=200$  that yields RMSE=0.8870. In order to combine the models, we split the predictions into 15 bins based on their support and compute a separate linear combination within each bin; see [3]. Such a combination leads to an RMSE of 0.8794. Finally we correct for a single day effect to achieve RMSE=0.8788.

A stronger correction accounts for periods longer than a single day, and also tries to characterize the recent user behavior on *similar* movies. To this end we compute pairwise similarities between all movies, denoted by  $s_{ij}$ , which are defined as the square of the Pearson correlation coefficient among the ratings of the two respective movies. Now, we weight the influence between movie  $i$  and all other movies rated by  $u$ . Those weights reflect both the similarity between the movies and the time proximity between the corresponding rating events, as follows:

$$w_{ij}^u = s_{ij} \cdot \exp(-\theta |t_{ui} - t_{uj}|)$$

Here, we are using  $\theta=0.2$ . Then we compute the following three magnitudes:

- $n_{ui} = \sum_{u \text{ rated } j} w_{ij}^u$
- $\bar{r}_{ui} = \frac{\sum_{u \text{ rated } j} w_{ij}^u \cdot r_{uj}}{\sum_{u \text{ rated } j} w_{ij}^u}$
- $V_{ui} = \frac{\sum_{u \text{ rated } j} w_{ij}^u \cdot (r_{uj})^2}{\sum_{u \text{ rated } j} w_{ij}^u} - (\bar{r}_{ui})^2$

As done previously, we use the weighted support and variance to compute a confidence coefficient:

- $c_{ui} = n_{ui} \cdot \exp(-\alpha \cdot V_{ui})$

Here, we use  $\alpha = 5$ . This way, the corrected score is:

$$\frac{\hat{r}_{ui} + c_{ui} \cdot \bar{r}_{ui}}{1 + c_{ui}}$$

We used this correction with three solutions in the final blend, as follows:

1. Post-process predictor #83 in last year's Progress Prize Report [3] to lower the RMSE from 0.9057 to 0.9037.
2. We applied 30 neighbors kNN on residuals of NSVD2 (200 factors) to obtain RMSE=0.8948. Then, by correcting the score the RMSE decreased to 0.8924.
3. We used a variant of the previously described neighborhood model, with mild temporal biases, as follows:

$$\hat{r}_{ui}(t) = \mu + b_u^{(1)}(t) + b_i(t) + |\mathcal{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}(u)} (r_{uj} - b_{uj}) w_{ij} + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} c_{ij}$$

The resulting RMSE of 0.8964 was improved to 0.8935 by applying the correction.

### 2.3.2 Blending multiple solutions

Our basic scheme for blending multiple predictors is based on a linear regression model as described in [3]. Also, occasionally we blend two predictors by partitioning the ratings into 15 bins based on user- and movie-support, to allow a separate linear combination within each bin [3]. This year we added new methods for blending predictors, to which we turn now.

Assume that we have a series of  $s$  predictors:  $r^{(k)} = \{r_{ui}^{(k)}\}_{u,i}$ ,  $k = 1, \dots, s$ . We would like to combine the  $s$  predictors into a blended predictor  $\hat{r}$ . Taking a simple linear combination turns into solving a regression problem seeking optimal values of the coefficients  $a^{(1)}, \dots, a^{(s)}$ , which will minimize the Probe RMSE of  $\hat{r} = \sum_{k=1}^s a^{(k)} \cdot r^{(k)}$ . However, such an

approach will assign a single, global weight to each predictor, without differentiating between the abilities of certain predictors to better model certain users or movies. Thus, we suggest introducing more coefficients: for each predictor  $k$  and movie  $i$ , we introduce the coefficient  $b_i^{(k)}$ . Likewise, for predictor  $k$  and user  $u$ , we introduce the coefficient  $c_u^{(k)}$ . Now, the combination of the  $s$  predictors is defined through:

$$\hat{r}_{ui} = \sum_{k=1}^s (a^{(k)} + b_i^{(k)} + c_u^{(k)}) \cdot r_{ui}^{(k)}$$

We train the model over the Probe set. The parameters are regularized, and globally optimal solution of the associated least squares problem can be obtained using a least squares solver. We used stochastic gradient descent (learning rate=  $2 \cdot 10^{-6}$ , weight decay=  $10^{-3}$ ) for the training.

Within the final blend, this scheme was used once. We combined two of last year's predictors. One was NNMF (60 factors) with adaptive user factors (RMSE=0.8973). The other was an RBM (100 hidden units; RMSE=0.9087). The combined predictor has an RMSE of 0.8871.

An issue with the above combination scheme is that it requires a separate set of parameters for each user, while in the Probe set (which is the training set in this context), there are very few ratings per user, making learning those parameters unreliable. In order to avoid this, we need to borrow information across users. One way to achieve this is by assuming that the user support (number of associated ratings in the full data set) determines the relative success of a single predictor on a user. Thus, we refrain from directly parameterizing users, but refer to them through their support. As for the movies, we have more information on them in the Probe data, so we still use a separate parameter per movie. Though, in order to borrow information across movies, we additionally address movies through their support.

Let  $n_u$  be the number of ratings associated with user  $u$  in the training data. We perform a log-transformation setting  $m_u = \log n_u$ . Finally, we center the resulting values, working with  $\widehat{m}_u$ . We follow the same procedure for movies: Let  $n_i$  be the number of ratings associated with movie  $i$  in the training data. We use a log-transformation setting  $m_i = \log n_i$ . Finally, we center the resulting values, working with  $\widehat{m}_i$ . The combination of the  $s$  predictors is defined as:

$$\hat{r}_{ui} = \sum_{k=1}^s \left( a^{(k)} + b_i^{(k)} + c^{(k)} \cdot \widehat{m}_i + d^{(k)} \cdot \widehat{m}_u \right) \cdot r_{ui}^{(k)}$$

The values of the parameters are learnt by stochastic gradient descent with weight decay on the Probe data.

This blending technique is used twice within the final blend:

1. We generate an RMSE=0.8771 predictor by combining four basic predictors:  
 (i) SimuFctr (60 factors; RMSE=0.9003), (ii) RBM (100 hidden units; RMSE=0.9087), (iii) 50 neighbors kNN on 100-unit RBM (RMSE=0.8888), (iv) SVD++<sup>(1)</sup> ( $f=200$ ; RMSE=0.8870).
2. We generate an RMSE=0.8855 predictor by combining five basic predictors, which were trained *without* including the Probe set in the training data even when generating the Quiz results: (i) SVD++<sup>(3)</sup> ( $f=50$ ; RMSE=0.8930), (ii) NNMF (60 factors; RMSE=0.9186), (iii) Integrated ( $f=100$ ,  $k=300$ ), (iv) RBM (100 hidden units; RMSE=0.9166), (v) GlobalNgbr ( $k=500$ ; RMSE=0.9125).

### **3. Older Methods**

Besides using the newer techniques described in the previous section, our solution also includes the following predictors that are based on techniques in the 2007-Progress Prize report [3]. In general, we believe that most of those techniques are inferior to the newly developed ones when considering both accuracy and efficiency.

#### **Asymmetric factor models**

1.  $rmse=0.9286$   
SIGMOID2 with k=40
2.  $rmse=0.9383$   
NSVD2 with k=40
3.  $rmse=0.9236$   
NSVD1 with k=200
4.  $rmse=0.9259$ ,  
NSVD1 with k=150
5.  $rmse=0.9260$   
NSVD1 with k=40
6.  $rmse=0.9225$   
SIGMOID1 with k=100

#### **Regression models**

7.  $rmse=0.9223$   
BIN-SVD3 based on 40 vectors
8.  $rmse=0.9212$   
PCA based on top 50 PCs
9.  $rmse=0.9241$   
PCA based on top 40 PCs
10.  $rmse=0.9335$   
BIN-SVD-USER based on 256 vectors
11.  $rmse=0.9290$   
BIN-SVD3-USER based on 65 vectors
12.  $rmse=0.9437$   
BIN-SVD-USER based on 196 vectors
13.  $rmse=0.9610$   
BIN-SVD-USER based on 100 vectors, but here we regressed residuals of double centering rather than the usual residuals of global effects
14.  $rmse=0.9414$   
BIN-SVD3-USER based on 40 vectors
15.  $rmse=0.9067$   
20 neighbors Corr-kNN on residuals of BIN-SVD-USER (60 vectors)
16.  $rmse=0.9030$   
50 neighbors kNN on residuals of BIN-SVD-USER (100 vectors)

- 17.  $rmse=0.9269$ ,  
PCA-USER, based on top 40 PCs
- 18.  $rmse=0.9302$ ,  
STRESS with 40 coordinates per movie

### **Restricted Boltzmann Machines with Gaussian visible units**

- 19.  $rmse=0.9052$   
800 hidden units
- 20.  $rmse=0.9044$   
400 hidden units
- 21.  $rmse=0.9056$   
256 hidden units
- 22.  $rmse=0.9429$   
100 hidden units, applied on raw data (no normalization/centering)
- 23.  $rmse=0.9074$   
100 hidden units, on residuals of full global effects
- 24.  $rmse=0.9267$   
256 hidden units, without conditional RBM, on residuals of full global effects

### **Restricted Boltzmann Machines**

We use conditional RBMs as described in [5].

- 25.  $rmse=0.9029$   
256-unit RBM
- 26.  $rmse=0.9029$   
200-unit RBM
- 27.  $rmse=0.9087$   
100-unit RBM
- 28.  $rmse=0.9093$   
100-unit RBM (learning rate=.15 decaying by 0.9 each iteration)

Using RBM as a pre-processor:

- 29.  $rmse=0.8960$   
Postprocessing residuals of 100-unit RBM with factorization
- 30.  $rmse=0.8905$   
50 neighbors kNN on 200-unit RBM
- 31.  $rmse=0.8904$   
40 neighbors kNN on 150-unit RBM

### **Matrix factorization**

- 32.  $rmse=0.8992$   
IncFctr (80 factors), adaptive user factors by [MseSim]
- 33.  $rmse=0.9070$   
[Corr-kNN] applied to residuals of SimuFctr (40 factors)

34.  $rmse=0.9050$   
     SimuFctr (40 factors), adaptive user factors with  $s_{ij} = \text{MSE}(i,j)^{-1/2}$
35.  $rmse=0.9026$   
     Cor-kNN on residuals of NNMF (60 factors)
36.  $rmse=0.8963$   
     NNMF (90 factors), adaptive user factors by [MseSim]
37.  $rmse=0.8986$   
     NNMF (90 factors), adaptive user factors by naive [SuppSim] (where  $x_{ij} = n_i \cdot n_j / n$ )
38.  $rmse=0.9807$   
     NNMF (90 factors), adaptive user factors by  $s_{ij} = \text{MSE}(i,j)^{-1/2}$
39.  $rmse=0.8970$   
     NNMF (90 factors), adaptive user factors by [SuppSim]
40.  $rmse=1.1561$   
     NNMF (128 factors), adaptive user factors by [MseSim], but adaptive user factors where computed with Lasso regularization, rather than Ridge regularization
41.  $rmse=0.9039$   
     NNMF (128 factors)
42.  $rmse=0.8955$ ,  
     NNMF (128 factors), adaptive user factors by [MseSim]
43.  $rmse=0.9072$   
     NNMF (60 factors)
44.  $rmse=0.9018$   
     NNMF (40 factors, adaptive user factors by [EditSim])
45.  $rmse=0.9426$   
     LassoNNMF (30 factors)
46.  $rmse=0.9327$   
     LassoNNMF (30 factors), adaptive user factors by [SuppSim]
47.  $rmse=0.8998$   
     Start with SimuFctr 60 factors, then a single GaussFctr iterations on movie side followed by many GaussFctr iterations on user side
48.  $rmse=0.9070$   
     Start with NNMF 90 factors, followed by many GaussFctr iterations on user side
49.  $rmse=0.9098$   
     Start with SimuFctr 40 factors, followed by many GaussFctr iterations on user side

### Neighborhood-based model (k-NN)

Some k-NN results were already mentioned. Here, we report the rest.

50.  $rmse=0.9309$   
     50 neighbors Fctr-kNN on residuals of full global effects. Weights based on 10 factors computed on binary matrix
51.  $rmse=0.9037$   
     75 neighbors Slow-kNN on residuals of SimuFctr (50 factors)

52.  $rmse=0.8953$   
     30 neighbors kNN on residuals of NNMF (180 factors)
53.  $rmse=0.9105$   
     50 neighbors kNN on residuals of all global effects except the last 4
54.  $rmse=0.9496$   
     25 neighbors kNN on raw scores (no normalization)
55.  $rmse=0.8979$   
     60 neighbors kNN on residuals of NNMF (60 factors)
56.  $rmse=0.9215$   
     50 neighbors Bin-kNN on residuals of full global effects, neighbor selection by  
     [CorrSim]
57.  $rmse=0.9097$   
     25 neighbors Fctr-kNN on residuals of NNMF (60 factors). Weights based on 10  
     NNMF factors
58.  $rmse=0.9290$   
     50 neighbors Fctr-kNN on raw scores. Weights based on 10 factors computed on  
     binary matrix
59.  $rmse=0.9097$   
     100 neighbors User-kNN on residuals of NNMF (60 factors)
60.  $rmse=0.9112$   
     100 neighbors User-kNN on residuals of SimuFctr (50 factors)
61.  $rmse=0.9248$   
     30 neighbors User-MSE-kNN on residuals of full global effects
62.  $rmse=0.9170$   
     Corr-kNN on residuals of full global effects
63.  $rmse=0.9079$   
     Corr-kNN on residuals of IncFctr (80 factors),
64.  $rmse=0.9237$   
     MSE-kNN on residuals of full global effects
65.  $rmse=0.9085$   
     Supp-kNN on residuals of SimuFctr (50 factors).
66.  $rmse=0.9110$   
     Supp-kNN on residuals of IncFctr (80 factors)
67.  $rmse=0.9440$   
     Supp-kNN on residuals of full global effects. Here, we used the more naïve  
     similarities where  $x_{ij} = n_i * n_j / n$
68.  $rmse=0.9335$   
     Supp-kNN on residuals of full global effects

### Combinations:

Each of the following results is based on mixing two individual results. Before mixing we split the user-movie pairs into 15 bins based on their support. For each bin we compute unique combination coefficients based on regression involving the Probe set.

69.  $rmse=0.8876$   
 Combination of [3]'s #36 with <NNMF (60 factors) adaptive user factors by MseSim>
70.  $rmse=0.8977$   
 Combination of #59 with #55
71.  $rmse=0.8906$   
 Combination of [3]'s #45 with [3]'s #73
72.  $rmse=0.9078$   
 Combination of #62 with <User-kNN on raw scores>
73.  $rmse=0.8967$   
 Combination of [3]'s #45 with [3]'s #50
74.  $rmse=0.8957$   
 Combination of [3]'s #45 with with <NNMF (60 factors) adaptive user factors by MseSim>
75.  $rmse=0.9017$   
 Combination of #53 with <User-kNN on residuals of all global effects except last 4>
76.  $rmse=0.8937$   
 Combination of [3]'s #45 with #54
77.  $rmse=0.8904$   
 Combination of [3]'s #45 with <30 neighbors kNN on residuals of SimuFctr (50 factors)>

### Imputation of Qualifying predictions:

We had predictions for the Qualifying set with RMSE of 0.8836. Then, we inserted the Qualifying set into the training set, while setting unknown scores to the RMSE= 0.8836 predictions. We tried some of our methods on this enhanced training set:

78.  $rmse=0.8952$   
 MSE-kNN on residuals of SimuFctr (20 factors)
79.  $rmse=0.9057$   
 SimuFctr (50 factors)
80.  $rmse=0.9056$   
 SimuFctr (20 factors), Probe set is excluded from training set
81.  $rmse=0.9093$   
 IncFctr (40 factors), adaptive user factors by [SuppSim]. Probe set is excluded from training set
82.  $rmse=0.9005$   
 MSE-kNN on residuals of IncFctr (40 factors)
83.  $rmse=0.9082$   
 50 neighbors kNN on residuals of global effects

### Specials:

84.  $rmse=1.1263$   
 Take binary matrix (rated=1, not-rated=0), and estimate it by 40 factors. Using these factors, construct predictions for the Probe and Qualifying set and center the

predictions for each set. Consequently, using the probe set we learn how to regress centered true ratings on these predictions, and do the same on the Qualifying set.

## 5. Discussion

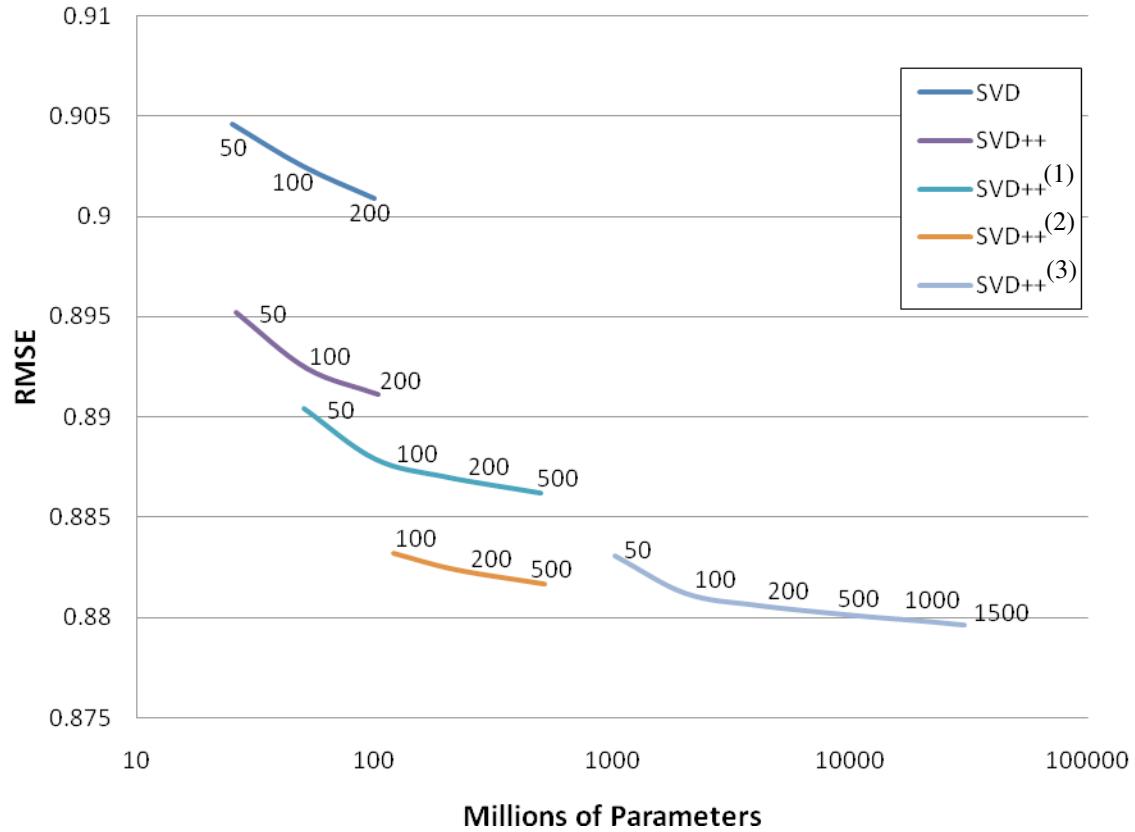
During the two years of analyzing the Netflix data, we have learnt several interesting lessons, which apparently are not reflected well in the prior literature. In the following we briefly discuss some of them.

Collaborative filtering methods address the sparse set of rating values. However, much accuracy is obtained by also looking at other features of the data. First is the information on which movies each user chose to rate, regardless of specific rating value (“the binary view”). This played a decisive role in our 2007 solution, and reflects the fact that the movies to be rated are selected deliberately by the user, and are not a random sample. Second important feature, which played a very significant role in our progress through 2008, is accounting for temporal effects and realizing that parameters describing the data are not static but dynamic functions. At the same time, we should mention that not all data features were found to be useful. For example, we tried to benefit from an extensive set of attributes describing each of the movies in the dataset. Those attributes certainly carry a significant signal and can explain some of the user behavior. However, we concluded that they could not help at all for improving the accuracy of well tuned collaborative filtering models.

Beyond selecting which features of the data to model, working with well designed models is also important. It seems that models based on matrix-factorization were found to be most accurate (and thus popular), as evident by recent publications and discussions on the Netflix Prize forum. We definitely agree to that, and would like to add that those matrix-factorization models also offer the important flexibility needed for modeling temporal effects and the binary view. Nonetheless, neighborhood models, which have been dominating most of the collaborative filtering literature, are still expected to be popular due to their practical characteristics - being able to handle new users/ratings without re-training and offering direct explanations to the recommendations. During our work we have found that the known heuristic-based neighborhood methods can be replaced with more profound methods (as those in Sec 2.2), which deliver much improved accuracy while retaining the useful properties of general neighborhood methods.

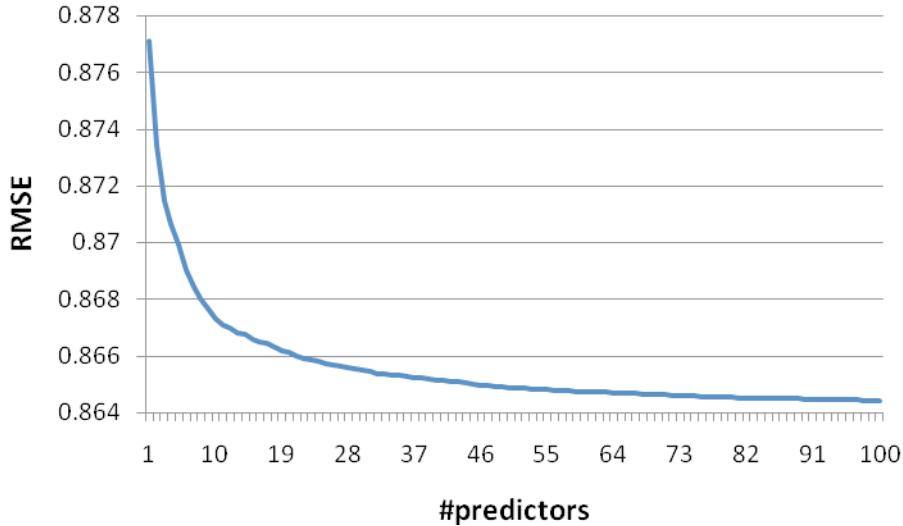
We were quite surprised by how many parameters can be added to a single model, while still improving prediction accuracy on the test set. In the chart below each curve corresponds to a matrix-factorization model, with an increasing number of parameters. It is evident that accuracy improves as we add more parameters to a single model, or as we move to models richer in parameters. Notice that accuracy improves even when fitting over 10 billion parameters to the dataset, which is somewhat surprising considering that the dataset contains just 100 million ratings. This indicates a complex multifaceted nature of user-movie interaction. We should remark that it is possible to build more memory

efficient models that achieve almost the same accuracy as the most complex models, but this is beyond the scope of this document.



**Figure 1.** Matrix factorization models – error vs. #parameters. The plot shows how the accuracy of each of five individual factor models improves by increasing the number of involved parameters (which is equivalent to increasing the dimensionality of the factor model, denoted by numbers on the charts). In addition, the more complex factor models, whose descriptions involve more distinct sets of parameters, are the more accurate ones.

Finally, using increasingly complex models is only one way of improving accuracy. An apparently easier way to achieve better accuracy is by blending multiple simpler models. The chart in Fig. 2 shows how the accuracy of our final solution improves with increasing the number of blended predictors. As expected, the first few predictors have a decisive contribution to improving accuracy, while the rest have a marginal contribution. A lesson here is that having lots of models is useful for the incremental results needed to win competitions, but practically, excellent systems can be built with just a few well-selected models.



**Figure 2.** The plot above shows RMSE as a function of the number of methods used. By blending five predictors one can achieve RMSE=0.8699. As more predictors are added accuracy slowly improves, till reaching RMSE=0.8643 with 100 predictors.

## References

1. R. Bell and Y. Koren, “Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights”, *IEEE International Conference on Data Mining (ICDM'07)*, IEEE, 2007.
2. R. Bell and Y. Koren, “Improved Neighborhood-based Collaborative Filtering”, *KDD-Cup and Workshop*, ACM press, 2007.
3. R. Bell and Y. Koren, and C. Volinsky, “The BellKor solution to the Netflix Prize”, [http://www.netflixprize.com/assets/ProgressPrize2007\\_KorBell.pdf](http://www.netflixprize.com/assets/ProgressPrize2007_KorBell.pdf), 2007.
4. Y. Koren, “Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model”, *Proc. 14th ACM Int. Conference on Knowledge Discovery and Data Mining (KDD'08)*, ACM press, 2008.
5. Y. Koren, “Factor in the Neighbors: Scalable and Accurate Collaborative Filtering”, <http://public.research.att.com/~volinsky/netflix/factorizedNeighborhood.pdf>, submitted.
6. A. Paterek, “Improving Regularized Singular Value Decomposition for Collaborative Filtering”, *KDD-Cup and Workshop*, ACM press, 2007.
7. G. Takacs, I. Pilaszy, B. Nemeth and D. Tikk, “Major Components of the Gravity Recommendation System”, *SIGKDD Explorations*, 9 (2007), 80-84.

## Appendix A. Combining with BigChaos

We combined our solution with the one produced by the BigChaos team in order to further improve accuracy. The combined solution, of RMSE=0.8616, is a linear combination of 207 predictor sets. Each predictor set was centered to have mean zero before the combination and an estimate of the Quiz set mean was added back after the combination. Any final predictions outside the range [1, 5] were clipped. The predictor sets include 100 BigChaos predictors, including neural net blends (see accompanying progress report document); 84 predictors described in the 2007 progress report (listed in Section 3 of this document), and 23 predictors described in Section 2 of this document.

Coefficients for the linear combination were computed via an approximate linear regression on the Quiz set. We utilize the fact that a linear regression requires knowing only sufficient statistics that can be estimated from RMSEs of the Quiz predictors and other known quantities. Due to the large number of blended predictors, many of which are close to collinear, some of the estimated coefficients are very unstable without regularization. Consequently, we used ridge regression, with ridge parameter (or, “regularization constant”) equaling 0.001. Computation is done by a standard least squares solver library (LAPACK).

# Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights\*

Robert M. Bell and Yehuda Koren

AT&T Labs – Research

180 Park Ave, Florham Park, NJ 07932

{rbell,yehuda}@research.att.com

## Abstract

*Recommender systems based on collaborative filtering predict user preferences for products or services by learning past user-item relationships. A predominant approach to collaborative filtering is neighborhood based (“ $k$ -nearest neighbors”), where a user-item preference rating is interpolated from ratings of similar items and/or users. We enhance the neighborhood-based approach leading to substantial improvement of prediction accuracy, without a meaningful increase in running time. First, we remove certain so-called “global effects” from the data to make the ratings more comparable, thereby improving interpolation accuracy. Second, we show how to simultaneously derive interpolation weights for all nearest neighbors, unlike previous approaches where each weight is computed separately. By globally solving a suitable optimization problem, this simultaneous interpolation accounts for the many interactions between neighbors leading to improved accuracy. Our method is very fast in practice, generating a prediction in about 0.2 milliseconds. Importantly, it does not require training many parameters or a lengthy preprocessing, making it very practical for large scale applications. Finally, we show how to apply these methods to the perceptively much slower user-oriented approach. To this end, we suggest a novel scheme for low dimensional embedding of the users. We evaluate these methods on the Netflix dataset, where they deliver significantly better results than the commercial Netflix Cinematch recommender system.*

## 1 Introduction

Recommender systems analyze patterns of user interest in items or products to provide personalized recommendations of items that will suit a user’s taste [1]. Increasingly, their excellent ability to characterize and recommend items within huge collections represent a computerized alternative to human recommendations. Because good personalized recommendations can add another dimension to

the user experience, e-commerce leaders like Amazon.com and online movie rental company Netflix, have made recommender systems a salient part of their web sites.

Broadly speaking, recommender systems use either of two strategies. The *content based approach* profiles each user or product, allowing programs to associate users with matching products. For example, a movie profile might describe its genre, the participating actors, its box office popularity, etc. User profiles could include demographic information or answers to a suitable questionnaire. Of course, content based strategies require gathering external information that might not be available or easy to collect.

We focus on an alternative strategy, known as *Collaborative Filtering* (CF) [6], which relies only on past user behavior—e.g., their previous transactions or product ratings—and does not require the creation of explicit profiles. CF analyzes relationships between users and interdependencies among products, in order to identify new user-item associations. Besides avoiding the need for extensive data collection about items or users, CF requires no domain knowledge. In addition, it offers the potential to uncover patterns that would be difficult or impossible to profile using content based techniques. This has led to many papers (e.g., [8]), research projects (e.g., [9]) and commercial systems (e.g., [11]) based on CF.

From a more abstract perspective, CF can be cast as missing value estimation. Given a user-item matrix of scores with many missing values, our goal is to estimate the missing values based on the given ones. The observed user-item scores measure the amount of interest between respective users and items. We call these user-item scores *ratings*, and they constitute the input to our algorithm.

The most common form of CF is the neighborhood-based approach (also known as “ $k$  Nearest Neighbors” or kNN, for short). The kNN methods identify pairs of items that tend to be rated similarly or like-minded users with similar histories of rating or purchasing, in order to predict ratings for unobserved users-item pairs.

Three major components characterize the kNN approach: (1) data normalization, (2) neighbor selection, and

\*An earlier version of this work appeared in *KDD Cup and Workshop*, ACM, 2007.

(3) determination of interpolation weights. Our experience has shown little difference among neighbor selection strategies (e.g., distance-based vs. correlation-based). However, the two other components, namely data normalization and interpolation weights, have proved vital to the success of the scheme. Accordingly, we revisit these two components and suggest novel methods to significantly improve the accuracy of kNN approaches without meaningfully affecting running time. Our three main contributions are:

1. It is customary to normalize the data before activating a kNN method. This brings different ratings to a closer level, which allows a better mixing thereof. Usually this is achieved by adjusting for the varying mean ratings across users and/or items. In Section 3 we offer a more comprehensive normalization that considers additional effects that are readily available in virtually all ratings datasets. This allows us to explain and eliminate much of the interfering variability from the data and to work with residuals that are more amenable to mutual interpolation.
2. Past kNN methods relate items (or users) by various heuristic variants of correlation coefficients, which allowed direct interpolation from neighbors' scores. In Section 4 we offer a rigorous alternative to these interpolation weights based on global optimization of a cost function pertaining to all weights simultaneously. This results in another improvement of estimation quality with a minor increase in running time.
3. The kNN approach can capitalize on two different kinds of information: item-item, how the subject user rated similar items; and user-user, how the subject item was rated by like-minded users. Because there typically are many more users than items in a system, the user-oriented approach is known as slower and less accurate. In Section 5 we discuss benefits of also employing a user-oriented approach, and we offer a novel method for alleviating the inherent computational difficulties of the user-oriented approach.

An encouraging evaluation of the results on the Netflix Prize user-movie dataset [3] is provided in Section 6.

## 2 Related Works

We are given ratings about  $m$  users and  $n$  items, arranged in an  $m \times n$  matrix  $R = \{r_{ui}\}_{1 \leq u \leq m, 1 \leq i \leq n}$ . We reserve special indexing letters for distinguishing users from items: for users  $u, v$ , and for items  $i, j, k$ .

### 2.1 Neighborhood-based collaborative filtering

The original neighborhood-based approach, which was shared by virtually all earlier CF systems, is user-oriented;

see [8] for a good analysis. In order to estimate the unknown rating  $r_{ui}$ , we resort to a set of *users*  $N(u; i)$  that tend to rate similarly to  $u$  ("neighbors"), and that actually rated item  $i$  (i.e.,  $r_{vi}$  is known for each  $v \in N(u; i)$ ).

The predicted value of  $r_{ui}$  is taken as a weighted average of the neighbors' ratings:

$$r_{ui} \leftarrow \frac{\sum_{v \in N(u; i)} s_{uv} r_{vi}}{\sum_{v \in N(u; i)} s_{uv}} \quad (1)$$

An analogous alternative to the user-oriented approach is the item-oriented approach [11, 15]. In those methods, to estimate the unknown  $r_{ui}$ , we identify a set of neighboring *items*  $N(i; u)$  that other users tend to rate similarly to their rating of  $i$ . All items in  $N(i; u)$  must have been rated by  $u$ . Then, in parallel to (1), the estimated value of  $r_{ui}$  is taken as a weighted average of the ratings of neighboring items:

$$r_{ui} \leftarrow \frac{\sum_{j \in N(i; u)} s_{ij} r_{uj}}{\sum_{j \in N(i; u)} s_{ij}} \quad (2)$$

The similarities—denoted by  $s_{uv}$  or  $s_{ij}$ —play a central role here, as they are used both for selecting the neighbors and for weighting the above averages. Common choices are the Pearson correlation coefficient and the closely related cosine similarity. Methods also differ by how they normalize or center data prior to activating interpolation rule (1) or (2). For example, correcting for user-specific means can improve prediction quality. We present a more comprehensive treatment of data normalization in Section 3. Sarwar et al. [15] found that item-oriented approaches deliver better quality estimates than user-oriented approaches while allowing more efficient computations. The greater efficiency occurs because, typically, the number of items is significantly lower than the number of users, which allows pre-computing all item-item similarities for retrieval as needed.

Neighborhood-based methods became very popular because they are intuitive and relatively simple to implement. In particular, they do not require tuning many parameters or an extensive training stage. They also provide a concise and intuitive justification for the computed predictions. This enables presenting the user a list of similar items that he or she has previously rated, as the basis for the estimated rating. This way, the user actually understands the reasoning behind the recommendation, allowing him/her to better assess its relevance (e.g., downgrade the estimated rating if it is based on an item that he or she no longer likes), or even encourage the user to alter outdated ratings.

However, standard neighborhood-based methods raise some concerns:

1. The similarity function ( $s_{uv}$  or  $s_{ij}$ ), which directly defines the interpolation weights, is arbitrary. Various CF algorithms use somewhat different similarity measures, trying to quantify the elusive notion of user- or

- item-similarity. Suppose that a particular item is predicted perfectly by a subset of the neighbors. In that case, we would want the predictive subset to receive all the weight, but that is impossible for bounded similarity scores like the Pearson correlation coefficient.
2. Previous neighborhood-based methods do not account for interactions among neighbors. Each similarity between an item  $i$  and a neighbor  $j \in N(i; u)$  is computed independently of the content of  $N(i; u)$  and the other similarities:  $s_{ik}$  for  $k \in N(i; u) - \{j\}$ . For example, suppose that our items are movies, and the neighbors set contains three movies that are highly correlated with each other (e.g., sequels such as “Lord of the Rings 1–3”). An algorithm that ignores the similarity of the three movies when determining their interpolation weights, may end up essentially triple counting the information provided by the group.
  3. By definition, the interpolation weights sum to one, which may cause overfitting. Suppose that an item has no useful neighbors rated by a particular user. In that case, it would be best to ignore the neighborhood information, staying with the current data normalization. Nevertheless, the standard neighborhood formula uses a weighted average of ratings for the uninformative neighbors.
  4. Neighborhood methods may not work well if variability differs substantially among neighbors.

In a recent work [2], we overcame most of these shortcomings, but the resulting algorithm was several orders of magnitude slower than previous neighborhood-methods, thereby limiting its practical applicability. In this work, we address the aforementioned issues of neighborhood based approaches, without compromising running time efficiency.

### 3 Normalizing by Removing Global Effects

Although neighborhood and factorization based CF are both powerful prediction methods, there are several reasons to precede either technique with simple models that estimate what we call “global effects.” First, there may be large user and item effects—i.e., systematic tendencies for some users to give higher ratings than other users and for some items to receive higher ratings than others. The basic kNN interpolation method detailed in equations (1)-(2) requires ratings where user and item effects have been taken out in order to, e.g., avoid predicting too high for low-rated items that happen to have a lot of neighbors with high average ratings, and vice versa.

Second, one may have access to information about either the items or users that can benefit the model. Although factorization offers the potential to detect such structure through estimation of latent variables, directly incorporating variables such as movie genre and user demographics

may be more effective and does not require training a factorization model. While such content based analysis is beyond the scope of this paper, we do consider two types of easily derived variables: the number of ratings of an item or by a user and the average rating of an item or by a user. Variables like these allow us, for example, to distinguish users who like the most commonly rated movies best from those who prefer more specialized fare (after controlling for both movie and user effects).

Third, there may be characteristics of specific ratings, such as the date of the rating, that explain some of the variation in scores. For example, a particular user’s ratings may slowly, or suddenly, rise over time, above and beyond any change explained by the inherent quality of the items being rated. Similarly, ratings for some movies may fall with time after their initial release dates, while others stand the test of time quite well. Neither factorization nor kNN could be expected to detect patterns like these.

#### 3.1 Methods

Our strategy is to estimate one “effect” at a time, in sequence (i.e., the main effect for items, the main effect for users, a user-time interaction, etc.). At each step, we use residuals from the previous step as the dependent variable for the current step. Consequently, after the first step, the  $r_{ui}$  refer to residuals, rather than raw ratings.

For each of the effects mentioned above, our goal is to estimate either one parameter for each item or one parameter for each user. For the rest of this subsection, we describe our methods for estimating user specific parameters; the method for items is perfectly analogous.

We denote by  $x_{ui}$  the explanatory variable of interest corresponding to user  $u$  and item  $i$ . For user main effects, the  $x_{ui}$ ’s are identically 1. For other global effects, we center  $x_{ui}$  for each user by subtracting the mean of  $x_{ui}$  for that user. In each case, our model is:

$$r_{ui} = \theta_u x_{ui} + \text{error} \quad (3)$$

With sufficient ratings for user  $u$ , we might use the unbiased estimator:

$$\hat{\theta}_u = \frac{\sum_i r_{ui} x_{ui}}{\sum_i x_{ui}^2}$$

where each summation is over all items rated by  $u$ . However, for sparse data, some values of  $\hat{\theta}_u$  may be based on very few observations, resulting in unreliable estimates.

To avoid overfitting, we shrink individual values of  $\hat{\theta}_u$  towards a common value. Shrinkage can be motivated from a Bayesian perspective. Suppose that the true  $\theta_u$  are independent random variables drawn from a normal distribution,

$$\theta_u \sim N(\mu, \tau^2)$$

for known  $\mu$  and  $\tau^2$ , while

$$\hat{\theta}_u | \theta_u \sim N(\theta_u, \sigma_u^2)$$

for known  $\sigma_u^2$ . We estimate  $\theta_u$  by its posterior mean:

$$E(\theta_u|\hat{\theta}_u) = \frac{\tau^2\hat{\theta}_u + \sigma_u^2\mu}{\tau^2 + \sigma_u^2}$$

a linear combination of the empirical estimator  $\hat{\theta}_u$  and the common mean  $\mu$ . The parameter  $\sigma_u^2$  can be estimated from the formula for the variance of a weighted mean, while  $\mu$  can be estimated by the mean of the  $\theta_u$ 's (optionally weighted by  $n_u$ ). Empirical Bayes [4] suggests that the maximum likelihood estimate of  $\tau^2$  can be found as the solution to:

$$\tau^2 = \frac{\sum_u[(\hat{\theta}_u - \mu)^2 - \sigma_u^2]/(\tau^2 + \sigma_u^2)^2}{\sum_u(\tau^2 + \sigma_u^2)^{-2}}$$

In practice, we used a slightly simpler estimator for  $\theta_u$  by assuming  $\mu = 0$  and  $\sigma_u^2$  is proportional to  $1/n_u$ , to yield:

$$\frac{n_u\hat{\theta}_u}{n_u + \alpha}$$

where  $n_u$  is the number of ratings by user  $u$  and  $\alpha$  is a constant. We determined  $\alpha$  by cross validation.

### 3.2 Example

We illustrate the impact of estimating successive global effects on the Netflix data [3]. The dataset is based on more than 100 million ratings of movies performed by anonymous Netflix customers and is described in details in Section 6. Here, we report results on the *Probe set*, which is a test set containing about 1.4 million ratings compiled by Netflix. Accuracy of predictions is measured by their root mean squared error (RMSE).

Table 1 shows how the RMSE for the probe set declines with the inclusion of each successive global effect. Not surprisingly, by far the largest improvements in RMSE are associated with the two sets of main effects for movies and for users. They reduce the RMSE from 1.1296 based on use of the mean rating in the training data, down to 0.9841.

Of more interest are various interactions. The first interaction term,  $\text{User} \times \text{Time}(\text{user})^{1/2}$ , allows each user's rating to change linearly with the square root of the number of days since the user's first rating. Exploratory analysis indicated that this transformation improved the fit relative to the untransformed number of days. Technically, we set each  $x_{ui}$  in (3) to the square root of the number of days elapsed since the first rating by user  $u$  till the time  $u$  rated item  $i$ . Then, for each fixed user  $u$ , we center all computed values of  $x_{ui}$ , and calculate  $\hat{\theta}_u$  in order to predict  $r_{ui}$  based on  $x_{ui}$ . This interaction, reduces the RMSE by 0.0032.

Similarly,  $\text{User} \times \text{Time}(\text{movie})^{1/2}$ , allows each user's rating to change linearly with the square root of the number of days since the movie's first rating by anyone. Somewhat surprisingly, this latter interaction contributes almost

| Effect                              | RMSE   | Improvement |
|-------------------------------------|--------|-------------|
| Overall mean                        | 1.1296 | NA          |
| Movie effect                        | 1.0527 | .0769       |
| User effect                         | 0.9841 | .0686       |
| User $\times$ Time(user) $^{1/2}$   | 0.9809 | .0032       |
| User $\times$ Time(movie) $^{1/2}$  | 0.9786 | .0023       |
| Movie $\times$ Time(movie) $^{1/2}$ | 0.9767 | .0019       |
| Movie $\times$ Time(user) $^{1/2}$  | 0.9759 | .0008       |
| User $\times$ Movie average         | 0.9719 | .0040       |
| User $\times$ Movie support         | 0.9690 | .0029       |
| Movie $\times$ User average         | 0.9670 | .0020       |
| Movie $\times$ User support         | 0.9657 | .0013       |

**Table 1. RMSE for Netflix probe data after adding a series of global effects to the model**

as much as the first. Two additional time interactions concentrate on the complementary movie viewpoint. That is, for each movie, they model how its ratings change with time. Once again, the time variables are either square root of days since first rating of the movie, or square root of days since first rating by the user.

The next two effects interact users with movie characteristics. We measure the popularity of a movie in two different ways: (1) its average rating; and (2) its support, which is the number of ratings associated with the movie. These effects—User  $\times$  Movie average and User  $\times$  Movie support—measure how users change their ratings based on the popularity of the movies. We try to estimate here how closely a user follows the public opinion, or perhaps the opposite, how contrarian is the user. The most effective interaction is between users and movie average, indicating that users varied in terms of how much they agree with the consensus.

Finally, we interact movies with user characteristics, regressing the ratings of each movie on the mean or support of the users. Although these interactions are more difficult to motivate, each contributes an additional modest decrease of the RMSE. Overall, the eight interactions combine to reduce the RMSE by 0.0184 to 0.9657.

## 4 A Neighborhood Relationships Model

In this section we assume that global effects have already been removed, so whenever we refer to a rating we actually mean the residual remaining after removing global effects from the original ratings data. However, our discussion here is completely general, so the following method applies whether global effects are removed or not.

We need to estimate the unknown rating by user  $u$  of item  $i$ , that is  $r_{ui}$ . As with all neighborhood-based methods, our first step is neighbor selection. In this section, we focus on an item-oriented method. Among all items rated by  $u$ , we select the  $K$  most similar to  $i$ ,  $N(i; u)$ , by using a similarity function such as the correlation coefficient, as described later in this section. Typical values of  $K$  lie in the

range of 20–50; see Section 6.

Given a set of neighbors  $N(i; u)$ , we need to compute *interpolation weights*  $\{w_{ij} | j \in N(i; u)\}$  that will enable the best prediction rule of the form:

$$r_{ui} \leftarrow \sum_{j \in N(i; u)} w_{ij} r_{uj} \quad (4)$$

For notational convenience assume that the  $K$  neighbors in  $N(i; u)$  are indexed by  $1, \dots, K$ .

We seek a formal computation of the interpolation weights, that stems directly from their usage within prediction rule (4). As explained earlier, it is important to derive all interpolation weights simultaneously to account for interdependencies among the neighbors. We achieve these goals by defining a suitable optimization problem.

To start, we consider a hypothetical dense case, where all users but  $u$  rated both  $i$  and *all* its neighbors in  $N(i; u)$ . In that case, we could learn the interpolation weights by modeling the relationships between item  $i$  and its neighbors through a least squares problem:

$$\min_w \sum_{v \neq u} \left( r_{vi} - \sum_{j \in N(i; u)} w_{ij} r_{vj} \right)^2 \quad (5)$$

Notice that the only unknowns here are the  $w_{ij}$ 's. The optimal solution to the least squares problem (5) is found by differentiation as a solution of a linear system of equations. From a statistics viewpoint, it is equivalent to the result of a linear regression (without intercept) of  $r_{vi}$  on the  $r_{vj}$  for  $j \in N(i; u)$ . Specifically, the optimal weights are given by:

$$Aw = b \quad (6)$$

Here,  $A$  is a  $K \times K$  matrix defined as:

$$A_{jk} = \sum_{v \neq u} r_{vj} r_{vk} \quad (7)$$

Similarly the vector  $b \in \mathbb{R}^K$  is given by:

$$b_j = \sum_{v \neq u} r_{vj} r_{vi} \quad (8)$$

For a sparse ratings matrix there are likely to be very few users who rated  $i$  and all its neighbors  $N(i; u)$ . Accordingly, it would be unwise to base  $A$  and  $b$  as given in (7)–(8) only on users with complete data. Even if there are enough users with complete data for  $A$  to be nonsingular, that estimate would ignore a large proportion of the information about pairwise relationships among ratings by the same user. However, we can still estimate  $A$  and  $b$ , up to the same constant, by averaging over the given pairwise support, leading to the following reformulation:

$$\bar{A}_{jk} = \frac{\sum_{v \in U(j, k)} r_{vj} r_{vk}}{|U(j, k)|} \quad (9)$$

$$\bar{b}_j = \frac{\sum_{v \in U(i, j)} r_{vj} r_{vi}}{|U(i, j)|} \quad (10)$$

where  $U(j, k)$  is the set of users who rated both  $j$  and  $k$ .

This is still not enough to overcome the sparseness issue. The elements of  $\bar{A}_{jk}$  or  $\bar{b}_j$  may differ by orders of magnitude in terms of the number of users included in the average. As discussed in the previous section, averages based on relatively low support (small values of  $|U(j, k)|$ ) can generally be improved by shrinkage towards a common mean. Thus, we compute a baseline value that is defined by taking the average of all possible  $\bar{A}_{jk}$  values. Let us denote this baseline value by  $avg$ ; its precise computation is described in the next subsection. Accordingly, we define the corresponding  $K \times K$  matrix  $\hat{A}$  and the vector  $\hat{b} \in \mathbb{R}^K$ :

$$\hat{A}_{jk} = \frac{|U(j, k)| \cdot \bar{A}_{jk} + \beta \cdot avg}{|U(j, k)| + \beta} \quad (11)$$

$$\hat{b}_j = \frac{|U(i, j)| \cdot \bar{b}_j + \beta \cdot avg}{|U(i, j)| + \beta} \quad (12)$$

The parameter  $\beta$  controls the extent of the shrinkage. A typical value on the Netflix data when working with residuals of full global effects is  $\beta = 500$ .

Our best estimate for  $A$  and  $b$  are  $\hat{A}$  and  $\hat{b}$ , respectively. Therefore, we modify (6) so that the interpolation weights are defined as the solution of the linear system:

$$\hat{A}w = \hat{b} \quad (13)$$

The resulting interpolation weights are used within (4) in order to predict  $r_{ui}$ . When working with residuals of global effects, they should be added back to the predicted  $r_{ui}$ , which completes the computation.

This method addresses all four concerns raised in Section 2.1. First, interpolation weights are derived directly from the ratings, not based on any similarity measure. Second, the interpolation weights formula explicitly accounts for relationships among the neighbors. Third, the sum of the weights is not constrained to equal one. If an item (or user) has only weak neighbors, the estimated weights may all be very small. Fourth, the method automatically adjusts for variations among items in their means or variances.

#### 4.1 Preprocessing

Efficient computation of an item-item neighborhood-based method requires precomputing certain values associated with each item-item pair for rapid retrieval.

First, we need a quick access to all item-item similarities, the  $s_{ij}$  values. These similarities are required for identifying the  $K$  neighbors that constitute  $N(i; u)$ . Here, we usually take  $s_{ij}$  as the Pearson correlation coefficient between

$i$  and  $j$  calculated on their common raters. It is important to shrink  $s_{ij}$  based on their support, e.g., multiplying the correlation by:  $|\mathcal{U}(i, j)| / (|\mathcal{U}(i, j)| + \alpha)$  for some small  $\alpha$ .

Second, we precompute all possible entries of  $\hat{A}$  and  $\hat{b}$ . To this end, for each two items  $i$  and  $j$ , we compute:

$$\bar{A}_{ij} = \frac{\sum_{v \in \mathcal{U}(i, j)} r_{vi} r_{vj}}{|\mathcal{U}(i, j)|}$$

Then, the aforementioned baseline value  $avg$ , which is used in (11)-(12), is taken as the average entry of the precomputed  $n \times n$  matrix  $\bar{A}$ . In fact, we recommend using two different baseline values, one by averaging the non-diagonal entries of  $\bar{A}$  and another one by averaging the generally-larger diagonal entries. Finally, we derive a full  $n \times n$  matrix  $\hat{A}$  from  $\bar{A}$  by (11), using the appropriate value of  $avg$ . Here, the non-diagonal average is used when deriving the non-diagonal entries of  $\hat{A}$ , whereas the diagonal average is used when deriving the diagonal entries of  $\hat{A}$ .

Because of symmetry, it is sufficient to store the values of  $s_{ij}$  and  $\hat{A}_{ij}$  only for  $i \geq j$ . We allocated one byte for each individual value, so the overall space required for  $n$  items is exactly  $n(n + 1)$  bytes. For example, for the Netflix dataset which contains 17770 movies, overall memory requirements are 300MB, easily fitting within core memory. A more comprehensive system that includes data on 100,000 items would require about 10GB of space, which can fit within core memory of current 64bit servers. For even larger systems, disk resident storage of item-item values is still practical, as evident by the Amazon item-item recommender system, which accesses stored similarity information for several million catalog items [11]. Preprocessing time is quadratic in  $n$  and linear in the number of ratings. The time required for computing all  $s_{ij}$ 's and  $\hat{A}_{ij}$ 's on the Netflix data (which contains 100 million ratings) was about 15 minutes on a Pentium 4 PC. Notice that preprocessing can be easily parallelized.

Precomputing all possible entries of matrix  $\hat{A}$  saves the otherwise lengthy time needed to construct entries on the fly. After quickly retrieving the relevant entries of  $\hat{A}$ , we can compute the interpolation weights by solving a  $K \times K$  system of equations. For typical values of  $K$  (between 20 and 50), this time is comparable to the time needed for computing the  $K$  nearest neighbors, which is common to all neighborhood-based approaches. Hence, while our method relies on a much more detailed computation of the interpolation weights compared to previous methods, it does not significantly increase running time; see Section 6.

## 4.2 Calculation of the weights

The interpolation weights can be computed by solving (13) using standard linear equations solvers. However, we experience a modest increase in prediction accuracy when  $w$  is constrained to be nonnegative. This requires a

```

NonNegativeQuadraticOpt( $A \in \mathbb{R}^{K \times K}, b \in \mathbb{R}^K$ )
% Minimize  $x^T Ax - 2b^T x$  s.t.  $x \geq 0$ 

do
   $r \leftarrow Ax - b$  % the residual, or “steepest gradient”
  % find active variables - those that are pinned due to
  % nonnegativity constraints; set respective  $r_i$ 's to zero
  for  $i = 1, \dots, k$  do
    if  $x_i = 0$  and  $r_i < 0$  then
       $r_i \leftarrow 0$ 
    end if
  end for
   $\alpha \leftarrow \frac{r^T r}{r^T A r}$  % max step size
  % adjust step size to prevent negative values:
  for  $i = 1, \dots, k$  do
    if  $r_i < 0$  then
       $\alpha \leftarrow \min(\alpha, -x_i/r_i)$ 
    end if
  end for
   $x \leftarrow x + \alpha r$ 
  while  $\|r\| < \epsilon$  % stop when residual is close to 0
  return  $x$ 

```

**Figure 1. Minimizing a quadratic function with non-negativity constraints**

quadratic program, which can be solved by calling “Non-NegativeQuadraticOpt( $\hat{A}, \hat{b}$ )”, as shown in Figure 1. The function is based on the principles of the Gradient Projection method; see, e.g., [12].

## 5 User-oriented computation

A user-oriented approach can be implemented by switching the roles of users and items throughout the derivation of the previous section. However, as with previous neighborhood-based approaches, an item-oriented approach enables a much faster implementation by precomputing and storing in main memory a full item-item matrix containing all  $s_{ij}$  and  $\hat{A}_{ij}$  values for future retrieval. Typically, the larger number of users complicates such a precomputation in terms of both time and space, and out-of-core storage is required. Moreover, our experience with the Netflix data, under various settings, showed that an item-oriented approach consistently delivered more accurate predictions than a user-oriented one.

Nonetheless, user-oriented approaches identify different kinds of relations that item-oriented approaches may fail to recognize, and thus can be useful on certain occasions. For example, suppose that we want to predict  $r_{ui}$ , but none of the items rated by user  $u$  is really similar to  $i$ . In this case, an item-oriented approach will face obvious difficulties. However, when employing a user-oriented perspective, we may find a set of users similar to  $u$ , who rated  $i$ . The ratings of  $i$

by these users would allow us to improve prediction of  $r_{ui}$ .

Another common case is when a user  $u$  has barely provided any ratings to the system, but we have a rich history of past transactions by him—e.g., his purchase history, viewed pages history, searched items and other kinds of implicit information. By employing a user-oriented method on the implicit information, we can relate user  $u$  to other users who did provide ratings for item  $i$ , thereby derive  $r_{ui}$ . Finally, we can improve predicton accuracy by mixing the results of the item-oriented approach with those of the user-oriented one. In this section, we present a way to efficiently capitalize on the additional information captured within user-user relationships.

### 5.1 Low dimensional embedding of the users

When applying our method in a user-oriented manner, the major computational bottleneck is the creation of  $N(u; i)$ , the  $K$  users most similar to  $u$  that rated item  $i$ . Here, the typically huge number of user-user similarities complicates their storage and precomputation. Moreover, because we expect many more users than items, scanning all users that rated  $i$  for picking the  $K$  most similar to  $u$  becomes a rather lengthy process. Hence, it is crucial to measure the similarity between users very efficiently.

We significantly lower the computational effort of measuring user-user similarities by embedding all users in a low dimensional Euclidean space. To this end, we view each user as a point within the space of  $n$  movies. A common way for reducing dimensionality is by using Principal Component Analysis (PCA), which is often carried out through the closely related Singular Value Decomposition (SVD). Recall that the  $n$ -dimensional user points are arranged within the  $m \times n$  matrix  $R$ . Consequently, SVD computes the best rank- $f$  approximation  $R^f$ , which is defined as the product of two rank- $f$  matrices  $P_{m \times f}$  and  $Q_{n \times f}$ , where  $\|R - PQ^T\|_F$  is minimized. In a way, the matrix  $P$  is an optimal  $f$ -dimensional embedding of the users.

Applying an SVD-based technique to CF raises unique difficulties due to the sparsity issue. The conventional SVD computation requires that all entries of  $R$  are known. In fact, the goal of SVD is not properly defined when some entries of  $R$  are missing. Previous works have adapted matrix factorizations techniques to handle missing data in a variety of ways [2, 5, 7, 13, 14]. For our special needs, we suggest the following approach.

Let us symbolize the set of  $(u, i)$  pairs for which  $r_{ui}$  is known by  $\mathcal{K}$ . The goal of SVD, when restricted to the known ratings, is to minimize:

$$\sum_{(u,i) \in \mathcal{K}} (r_{ui} - p_u^T q_i)^2 \quad (14)$$

Here,  $p_u$  is the  $u$ -th row of  $P$ , which corresponds to user  $u$ . Likewise,  $q_i$  is the  $i$ -th row of  $Q$ , which corresponds to

item  $i$ . A critical issue is to avoid overfitting for items and users with relatively sparse data. To this end we regularize the model by penalizing the norm of each  $p_u$  and  $q_i$  (“ridge regression”). That is, we replace (14) with:

$$\sum_{(u,i) \in \mathcal{K}} (r_{ui} - p_u^T q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2) \quad (15)$$

A typical choice of  $\lambda$  for the Netflix data is 0.3. To minimize (15), we employ an alternating least squares strategy. That is, we alternate between fixing  $Q$  and  $P$ , thereby obtaining a series of efficiently solvable least squares problems. To further avoid overfitting, we restrict all entries of  $P$  and  $Q$  to be nonnegative by using a *nonnegative least squares* solver [10]. In essence,  $P$  and  $Q$  form a regularized nonnegative matrix factorization for the partially observed matrix  $R$ .

When recomputing  $P$ , we address each user  $u$  as follows. Recall that  $n_u$  is the number of ratings by  $u$ . We use the  $n_u \times f$  matrix  $Q[u]$  to denote the restriction of  $Q$  to the items rated by  $u$ , and let the vector  $r_u \in \mathbb{R}^{n_u}$  contain the given ratings by  $u$  ordered as in  $Q[u]$ . The new value for  $p_u$  is given by solving a nonnegative least-squares problem:

$$\begin{pmatrix} Q[u] \\ \Lambda \end{pmatrix} p_u = \begin{pmatrix} r_u \\ 0 \end{pmatrix}$$

Here,  $\Lambda$  is an  $n_u \times n_u$  diagonal matrix, where all diagonal entries are  $\lambda n_u$ . The computation of  $Q$  is analogous.

Finally, following a few tens of iterations of recomputing  $P$  and  $Q$ , we converge at an  $f$ -dimensional embedding of the users -  $P$ . We use  $f = 10$ . Of course, the low dimensional embedding of the users is performed only once, at the preprocessing stage. Since then, whenever we need to find neighboring users, we do so in their low dimensional representation. This significantly alleviates the computational complexity of the user-based computations and facilitates their use in real life, large datasets, such as the Netflix data. Further performance gains can be achieved by organizing the  $f$ -dimensional user points within a space-partitioning data structure (such as a *kd-tree*), which allows very efficient retrieval of the nearest points (most similar users).

After identifying the  $K$  most similar users,  $N(u; i)$ , we continue with the usual computation of interpolation weights. Unlike the movie-oriented case, the sheer number of users prevents us from precomputing their inner products. However, at this stage we need to deal with only  $K$  users, which is not a major bottleneck in the process, especially when considering that individual users are typically associated with far fewer ratings compared to individual items. We provide experimental results in Subsection 6.3.

### 5.2 Unifying user- and item-relations in a single model

Our current model assumes that relationships among users are fixed across all items. However, in reality, a user  $v$

may be very predictive of user  $u$  for certain kinds of items, but far less predictive for other items. When predicting  $r_{ui}$ , we would like to derive user-user interpolation weights that reflect how the neighboring users relate to  $u$  with respect to the given item –  $i$ . Thus, when learning the interpolation weights, we give a higher weight to items similar to  $i$ , which may serve as a better proxy for the sought user relations. To this end, we introduce item-item similarities ( $s_{ij}$ ) into the user version of (5), which becomes:

$$\min_w \sum_{j \neq i} s_{ij} \left( r_{uj} - \sum_{v \in N(u; i)} w_{uv} r_{vj} \right)^2 \quad (16)$$

Similarly, the user-version of matrix  $A$  and vector  $b$  of (7)-(8), becomes:  $A_{v_1 v_2} = \sum_{j \neq i} s_{ij} r_{v_1 j} r_{v_2 j}$ ,  $b_v = \sum_{j \neq i} s_{ij} r_{v_j} r_{uj}$ . Essentially, these modifications inject item-item relationships into the user-user model. Possible choices for  $s_{ij}$  are the absolute value of the Pearson correlation coefficient, or an inverse of the squared error. As usual with item-item magnitudes, all  $s_{ij}$ 's can be precomputed and stored, so introducing them into the user-user model barely affects running time while benefiting prediction accuracy. Naturally, this is part of our default setting.

A parallel idea can be used for integrating user-awareness into the item-oriented model. However, it requires the item-item inner products to be computed specifically for each query in order to reflect the relevant user similarities. This prevents the precomputation and storage of all item-item inner products. Because items are typically associated with large numbers of ratings, an online computation of their inner products is very expensive and impractical for large datasets. Hence, currently, we are not adding user-awareness into the item-oriented models.

## 6 Experimental Study

We evaluated our algorithms on the Netflix data of more than 100 million movie ratings [3]. We are not aware of any publicly available CF dataset that is close to the scope and quality of this dataset.

To maintain compatibility with results published by others, we adopted some standards that were set by Netflix, as follows. First, quality of the results is measured by their root mean squared error (RMSE), a measure that puts more emphasis on large errors compared with the alternative of mean absolute error. In addition, we report results on two test sets compiled by Netflix, one is known as *the Probe set* and the other is known as *the Quiz set*. These two test sets were constructed similarly, with both containing about 1.4 million of the most recent movie ratings (by date) performed by the users. The true ratings of the Probe set are provided. However, we do not know the true ratings for the Quiz set, which are held by Netflix being the subject of the Netflix Prize contest. Netflix provides RMSE values to

competitors that submit their predictions for the Quiz set. The benchmark is Netflix's proprietary CF system, Cine-match, which achieved a RMSE of 0.9514 on this Quiz set. The two test sets contain many more ratings by users that do not rate much and are harder to predict. In a way, they represent real requirements from a CF system, which needs to predict new ratings from older ones, and to equally address all users, not only the heavy raters.

### 6.1 Experiments with the Probe set

Table 2 compares our item-oriented method to a kNN method that takes interpolation weights as Pearson correlation coefficients (shrunk to improve results), which represent the common approach to the problem. Both methods use the same item neighborhoods. Reported results are RMSE values computed on the Probe set. We apply the methods after varying stages of preprocessing the data. First, we applied the methods to the raw data, where scores were not normalized at all. Then, we removed the first two effects: the user effect and the item effect, which is similar to double-centering. When no neighborhood-interpolation is performed, such a normalization alone delivers a RMSE of 0.9841, as shown in the second column of the table. A more substantial normalization is achieved by accounting for all the global effects, as described in Section 3. Global effects on their own lower the Probe RMSE to 0.9657, before applying any neighborhood interpolation.

Finally, our method can be combined with a factorization approach, which models the ratings matrix through a low rank factorization [2, 5, 7, 13, 14]. In a way, factorization complements the more localized neighborhood-based methods by providing a higher level perspective to the ratings data. Notice that unlike the previously mentioned normalizations, factorization requires an extensive training in order to learn the factors. Our implementation delivered a RMSE of 0.9167 on the Probe set, before considering any neighborhood information. A kNN method can be used to improve factorization results, by applying it to the residuals remaining after subtracting the estimates generated by factorization. Effectively, error is smoothed by considering local information that the rather regional factorization approaches tend to miss.

The remaining columns of Table 2 contain results using the two methods for determining interpolation weights. We provide results representing the spectrum of viable choices for neighborhood size –  $K$ . For each value of  $K$ , the same neighborhoods are used for the two methods; the only difference lies in the determination of interpolation weights. The main findings are as follows.

- The jointly derived interpolation weights proposed in this paper uniformly outperformed standard correlation based weights. The improvement was 0.0071 when applied after factorization, but was much larger

| Data normalization | No interpolation<br>( $k = 0$ ) | Correlation-based interpolation |          |          | Jointly derived interpolation |          |          |
|--------------------|---------------------------------|---------------------------------|----------|----------|-------------------------------|----------|----------|
|                    |                                 | $k = 20$                        | $k = 35$ | $k = 50$ | $k = 20$                      | $k = 35$ | $k = 50$ |
| none (raw scores)  | NA                              | 0.9947                          | 1.002    | 1.0085   | 0.9536                        | 0.9596   | 0.9644   |
| double centering   | 0.9841                          | 0.9431                          | 0.9470   | 0.9502   | 0.9216                        | 0.9198   | 0.9197   |
| global effects     | 0.9657                          | 0.9364                          | 0.9390   | 0.9413   | 0.9194                        | 0.9179   | 0.9174   |
| factorization      | 0.9167                          | 0.9156                          | 0.9142   | 0.9142   | 0.9071                        | 0.9071   | 0.9071   |

**Table 2. Comparing our interpolation scheme against conventional correlation-based interpolation, by reporting RMSEs on the Probe set. Various levels of data normalization (preprocessing) are shown, and different sizes of item-neighborhoods ( $K$ ) are considered.**

with less extensive data normalizations.

- Use of the neighborhood model with our proposed interpolation weights substantially improved the predictions across all the data normalizations. The neighborhood approach reduced RMSE by 0.0096 even after the much more time consuming factorization method. Notably, the correlation-based neighborhood model produced a much more modest improvement of 0.0025 relative to factorization.
- The best neighborhood size  $K$  varied depending on the extent of prior data normalization. Interestingly, we found no difference in RMSE over the range 20 to 50 neighbors for the best combination—jointly derived interpolation weights after factorization.
- Even with incorporation of neighborhood information, more complete data normalization improved predictions, but with diminishing returns. For example, while global effects reduced RMSE by 0.0184 before the neighborhood model, only 0.0023 survived after incorporation of the neighborhood model.

The correlation based method required about 200 seconds to process the whole Probe set regardless of the value of  $K$ . Our method, with  $K = 20$ , increased the time slightly to about 235 seconds for the whole Probe set, where the added time represents the effort needed to solve the  $K \times K$  least squares problem for each query. Not surprisingly, increasing  $K$  further raised running times to around 355 seconds ( $K = 35$ ) or 470 seconds ( $K = 50$ ), due to the growing effort needed to solve the larger least squares problems. Even so, the slower mode of our method ( $K = 50$ ) processed a single rating query in less than 0.4 millisecond.

## 6.2 Experiments with the Quiz set

When computing predictions for the Quiz set, the test set held by Netflix, we subsume the Probe set into our training data. This makes Quiz results better than the Probe results. Accordingly, our item-oriented method produced a RMSE of 0.9086 (4.5% improvement over Netflix system) for the quiz set when applied to residuals of global effects. When applied to residuals of factorization the RMSE dropped to 0.8982 (5.6% improvement over Netflix system), at the cost of requiring the training of a factorization model.

These results are comparable to recently reported results for the Netflix data by methods that require an extensive training phase [2, 13]. Moreover, when mixed with other methods, results can be substantially improved. For example, an alternative method to introducing neighborhood-awareness into factorization-based results was described in [2]. While the accuracy of the two methods is similar, they differ considerably in their nature, and thus produce different ratings. Thus, they can be mixed effectively to produce a more accurate solution. Their mixtures achieve solutions with RMSEs around 0.8900 (6.45% improvement over Netflix system), which would currently rank high on the Netflix Prize Leaderboard<sup>1</sup>, even before mixing with the user-oriented approach to which we now turn.

## 6.3 Experiments with the user-oriented approach

We evaluated the user-oriented approach on the Quiz set. In contrast to the item-oriented method, which requires around 30–50 neighbors to maximize prediction accuracy, the user-oriented approach benefits from a larger number of neighbors. Our default setting uses 100 neighbors. This probably reflects the reality that the data contains many more users than movies, but individual users provide a relatively small amount of information. The low dimensional embedding of the users was essential for dealing with the Netflix data, by allowing prediction of a single rating in around 40 milliseconds. Though this is much slower than the movie-oriented computation, it is still considerably faster than the common full-dimensional computation, which requires about 400 milliseconds per prediction. Moreover, as hinted in Subsection 5.1, a more efficient utilization of the low dimensional embedding would involve a space-partitioning data structure in order to further speed up the neighborhood retrieval. Computation of the 10-dimesnisonal embedding itself required about 40 minutes.

While a low dimensional embedding of users allowed a 10X speedup compared with the full high-dimensional computation, it did not reduce prediction accuracy. The RMSE when employing the method on normalized data (after global effects were removed) is 0.9157. Surprisingly,

<sup>1</sup>[www.netflixprize.com/leaderboard](http://www.netflixprize.com/leaderboard)

this compares favorably, with a RMSE of 0.9174 achieved through a much longer run that computed neighborhoods on the original data. Hence, we deduce that dimensionality reduction is an effective tool for performing user-oriented methods.

While a RMSE of 0.9157 presents a 3.75% improvement over Netflix Cinematch result, it is still worse than our reported 0.9086 RMSE using the item-oriented method. However, since the two methods capture different sorts of information, we can get a better result by combining them. For example by taking a weighted average of the results (60% movie, 40% user), the RMSE was lowered to 0.9030. More sophisticated combinations might go beyond using a fixed ratio, to modify the ratio across queries by estimating the relative success of the user-oriented method.

## 7 Summary

Collaborative filtering through neighborhood-based interpolation (“kNN”) is probably the most popular way to create a recommender system. The success of these methods depends on the choice of the interpolation weights, which are used to estimate unknown ratings from neighboring known ones. Nevertheless, the literature lacks a rigorous way to derive these weights. In this work we showed how the interpolation weights can be computed as a global solution to an optimization problem that precisely reflects their role. Comparison with past kNN methods on the Netflix data, demonstrated a significant improvement of prediction accuracy without a meaningful increase in running time.

Our second, complementary contribution is related to data normalization. Normalization is essential to kNN methods, as otherwise mixing ratings pertaining to different unnormalized users or items can produce inferior results. This work offered a comprehensive approach to data normalization, fitting 10 effects that can be readily observed in user-item rating data. Those effects cause substantial data variability and mask the fundamental relationships between ratings. Hence, their inclusion brings the ratings closer and facilitates improved estimation accuracy.

A kNN method can be most effectively employed in an item-oriented manner, by analyzing relationships between items. However, certain cases call for a user-oriented approach. Examples include users that did not rate much or at all, or when it is needed to improve prediction accuracy by considering also user-user relationships. We show how to apply our method for the user-oriented case. Importantly, we offer a novel implementation, based on a low dimensional embedding of the users, that allows a significant computational speedup.

## References

- [1] G. Adomavicius and A. Tuzhilin, “Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions”, *IEEE Transactions on Knowledge and Data Engineering* **17** (2005), 634–749.
- [2] R. M. Bell, Y. Koren and C. Volinsky, “Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems”, *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.
- [3] J. Bennet and S. Lanning, “The Netflix Prize”, KDD Cup and Workshop, 2007.
- [4] B. Efron and C. Morris, “Data analysis using Stein’s estimator and its generalization”, *Journal American Statistical Association* **70** (1975), 311–319.
- [5] S. Funk, “Netflix Update: Try This At Home”, [sifter.org/~simon/journal/20061211.html](http://sifter.org/~simon/journal/20061211.html), 2006.
- [6] D. Goldberg, D. Nichols, B. M. Oki and D. Terry, “Using Collaborative Filtering to Weave an Information Tapestry”, *Communications of the ACM* **35** (1992), 61–70.
- [7] K. Goldberg, T. Roeder, D. Gupta and C. Perkins, “Eigentaste: A Constant Time Collaborative Filtering Algorithm”, *Information Retrieval* **4** (2001), 133–151.
- [8] J. L. Herlocker, J. A. Konstan, A. Borchers and John Riedl, “An Algorithmic Framework for Performing Collaborative Filtering”, *Proc. 22nd ACM SIGIR Conference on Information Retrieval*, pp. 230–237, 1999.
- [9] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon and J. Riedl, “GroupLens: Applying Collaborative Filtering to Usenet News”, *Communications of the ACM* **40** (1997), 77–87, [www.grouplens.org](http://www.grouplens.org).
- [10] C.L. Lawson and B. J. Hanson, *Solving Least Squares Problems*, Prentice-Hall, 1974.
- [11] G. Linden, B. Smith and J. York, “Amazon.com Recommendations: Item-to-item Collaborative Filtering”, *IEEE Internet Computing* **7** (2003), 76–80.
- [12] J. Nocedal and S. Wright, *Numerical Optimization*, Springer, 1999.
- [13] R. Salakhutdinov, A. Mnih, and G. Hinton, “Restricted Boltzmann Machines for Collaborative Filtering”, *Proc. 24th Annual International Conference on Machine Learning*, 2007.
- [14] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, “Application of Dimensionality Reduction in Recommender System – A Case Study”, *WEBKDD’2000*.
- [15] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, “Item-based Collaborative Filtering Recommendation Algorithms”, *Proc. 10th International Conference on the World Wide Web*, pp. 285–295, 2001.

## **Collaborative Filtering Recommender Systems**

By Michael D. Ekstrand, John T. Riedl  
and Joseph A. Konstan

### **Contents**

---

|          |   |            |
|----------|---|------------|
| <b>1</b> | <b>Introduction</b>                     | <b>82</b>  |
| 1.1      | History of Recommender Systems          | 84         |
| 1.2      | Core Concepts, Vocabulary, and Notation | 85         |
| 1.3      | Overview                                | 87         |
| <b>2</b> | <b>Collaborative Filtering Methods</b>  | <b>88</b>  |
| 2.1      | Baseline Predictors                     | 89         |
| 2.2      | User–User Collaborative Filtering       | 91         |
| 2.3      | Item–Item Collaborative Filtering       | 95         |
| 2.4      | Dimensionality Reduction                | 101        |
| 2.5      | Probabilistic Methods                   | 107        |
| 2.6      | Hybrid Recommenders                     | 111        |
| 2.7      | Selecting an Algorithm                  | 112        |
| <b>3</b> | <b>Evaluating Recommender Systems</b>   | <b>114</b> |
| 3.1      | Data Sets                               | 115        |
| 3.2      | Offline Evaluation Structure            | 116        |
| 3.3      | Prediction Accuracy                     | 117        |
| 3.4      | Accuracy Over Time                      | 119        |

|          |  |            |
|----------|--|------------|
| 3.5      | Ranking Accuracy                             | 120        |
| 3.6      | Decision Support Metrics                     | 122        |
| 3.7      | Online Evaluation                            | 125        |
| <b>4</b> | <b>Building the Data Set</b>                 | <b>128</b> |
| 4.1      | Sources of Preference Data                   | 129        |
| 4.2      | Rating Scales                                | 132        |
| 4.3      | Soliciting Ratings                           | 134        |
| 4.4      | Dealing with Noise                           | 136        |
| <b>5</b> | <b>User Information Needs</b>                | <b>138</b> |
| 5.1      | User Tasks                                   | 139        |
| 5.2      | Needs for Individual Items                   | 140        |
| 5.3      | Needs for Sets of Items                      | 141        |
| 5.4      | Systemic Needs                               | 144        |
| 5.5      | Summary                                      | 149        |
| <b>6</b> | <b>User Experience</b>                       | <b>150</b> |
| 6.1      | Soliciting Ratings                           | 150        |
| 6.2      | Presenting Recommendations                   | 151        |
| 6.3      | Recommending in Conversation                 | 153        |
| 6.4      | Recommending in Social Context               | 154        |
| 6.5      | Shaping User Experience with Recommendations | 157        |
| <b>7</b> | <b>Conclusion and Resources</b>              | <b>159</b> |
| 7.1      | Resources                                    | 162        |
|          | <b>References</b>                            | <b>164</b> |

## Collaborative Filtering Recommender Systems

Michael D. Ekstrand<sup>1</sup>, John T. Riedl<sup>2</sup>  
and Joseph A. Konstan<sup>3</sup>

*University of Minnesota, 4-192 Keller Hall, 200 Union St., Minneapolis,  
MN 55455, USA*

<sup>1</sup>[ekstrand@cs.umn.edu](mailto:ekstrand@cs.umn.edu); <sup>2</sup>[riedl@cs.umn.edu](mailto:riedl@cs.umn.edu); <sup>3</sup>[konstan@cs.umn.edu](mailto:konstan@cs.umn.edu)

### Abstract

Recommender systems are an important part of the information and e-commerce ecosystem. They represent a powerful method for enabling users to filter through large information and product spaces. Nearly two decades of research on collaborative filtering have led to a varied set of algorithms and a rich collection of tools for evaluating their performance. Research in the field is moving in the direction of a richer understanding of how recommender technology may be embedded in specific domains. The differing personalities exhibited by different recommender algorithms show that recommendation is not a one-size-fits-all problem. Specific tasks, information needs, and item domains represent unique problems for recommenders, and design and evaluation of recommenders needs to be done based on the user tasks to be supported. Effective deployments must begin with careful analysis of prospective users and their goals. Based on this analysis, system designers have a host of options for the choice of algorithm and for its embedding in the surrounding user experience. This paper discusses a wide variety of the choices available and their implications, aiming to provide both practitioners and researchers with an introduction to the important issues underlying recommenders and current best practices for addressing these issues.

# 1

---

## Introduction

---

Every day, we are inundated with choices and options. What to wear? What movie to rent? What stock to buy? What blog post to read? The sizes of these decision domains are frequently massive: Netflix has over 17,000 movies in its selection [15], and Amazon.com has over 410,000 titles in its Kindle store alone [7]. Supporting discovery in information spaces of this magnitude is a significant challenge. Even simple decisions — what movie should I see this weekend? — can be difficult without prior direct knowledge of the candidates.

Historically, people have relied on recommendations and mentions from their peers or the advice of experts to support decisions and discover new material. They discuss the week’s blockbuster over the water cooler, they read reviews in the newspaper’s entertainment section, or they ask a librarian to suggest a book. They may trust their local theater manager or news stand to narrow down their choices, or turn on the TV and watch whatever happens to be playing.

These methods of recommending new things have their limits, particularly for information discovery. There may be an independent film or book that a person would enjoy, but no one in their circle of acquaintances has heard of it yet. There may be a new indie band in another city whose music will likely never cross the local critic’s

radar. Computer-based systems provide the opportunity to expand the set of people from whom users can obtain recommendations. They also enable us to mine users' history and stated preferences for patterns that neither they nor their acquaintances identify, potentially providing a more finely-tuned selection experience.

There has been a good deal of research over the last 20 years on how to automatically recommend things to people and a wide variety of methods have been proposed [1, 140]. Recently, the *Recommender Systems Handbook* [122] was published, providing in-depth discussions of a variety of recommender methods and topics. This survey, however, is focused primarily on *collaborative filtering*, a class of methods that recommend items to users based on the preferences other users have expressed for those items.

In addition to academic interest, recommendation systems are seeing significant interest from industry. Amazon.com has been using collaborative filtering for a decade to recommend products to their customers, and Netflix valued improvements to the recommender technology underlying their movie rental service at \$1M via the widely-publicized Netflix Prize [15].

There is also a growing interest in problems surrounding recommendation. Algorithms for understanding and predicting user preferences do not exist in a vacuum — they are merely one piece of a broader user experience. A recommender system must interact with the user, both to learn the user's preferences and provide recommendations; these concerns pose challenges for user interface and interaction design. Systems must have accurate data from which to compute their recommendations and preferences, leading to work on how to collect reliable data and reduce the noise in user preference data sets. Users also have many different goals and needs when they approach systems, from basic needs for information to more complex desires for privacy with regards to their preferences.

In his keynote address at the 2009 ACM Conference on Recommender Systems, Martin [90] argued that the algorithms themselves are only a small part of the problem of providing recommendations to users. We have a number of algorithms that work fairly well, and while there is room to refine them, there is much work to be done on

user experience, data collection, and other problems which make up the whole of the recommender experience.

## 1.1 History of Recommender Systems

The capacity of computers to provide recommendations was recognized fairly early in the history of computing. Grundy [123], a computer-based librarian, was an early step towards automatic recommender systems. It was fairly primitive, grouping users into “stereotypes” based on a short interview and using hard-coded information about various stereotypes’ book preferences to generate recommendations, but it represents an important early entry in the recommender systems space.

In the early 1990s, collaborative filtering began to arise as a solution for dealing with overload in online information spaces. Tapestry [49] was a manual collaborative filtering system: it allowed the user to query for items in an information domain, such as corporate e-mail, based on other users’ opinions or actions (“give me all the messages forwarded by John”). It required effort on the part of its users, but allowed them to harness the reactions of previous readers of a piece of correspondence to determine its relevance to them.

Automated collaborative filtering systems soon followed, automatically locating relevant opinions and aggregating them to provide recommendations. GroupLens [119] used this technique to identify Usenet articles which are likely to be interesting to a particular user. Users only needed to provide ratings or perform other observable actions; the system combined these with the ratings or actions of other users to provide personalized results. With these systems, users do not obtain any direct knowledge of other users’ opinions, nor do they need to know what other users or items are in the system in order to receive recommendations.

During this time, recommender systems and collaborative filtering became an topic of increasing interest among human–computer interaction, machine learning, and information retrieval researchers. This interest produced a number of recommender systems for various domains, such as Ringo [137] for music, the BellCore Video Recommender [62] for movies, and Jester [50] for jokes. Outside of computer

science, the marketing literature has analyzed recommendation for its ability to increase sales and improve customer experience [10, 151].

In the late 1990s, commercial deployments of recommender technology began to emerge. Perhaps the most widely-known application of recommender system technologies is Amazon.com. Based on purchase history, browsing history, and the item a user is currently viewing, they recommend items for the user to consider purchasing.

Since Amazon’s adoption, recommender technology, often based on collaborative filtering, has been integrated into many e-commerce and online systems. A significant motivation for doing this is to increase sales volume — customers may purchase an item if it is suggested to them but might not seek it out otherwise. Several companies, such as NetPerceptions and Strands, have been built around providing recommendation technology and services to online retailers.

The toolbox of recommender techniques has also grown beyond collaborative filtering to include content-based approaches based on information retrieval, bayesian inference, and case-based reasoning methods [132, 139]. These methods consider the actual content or attributes of the items to be recommended instead of or in addition to user rating patterns. Hybrid recommender systems [24] have also emerged as various recommender strategies have matured, combining multiple algorithms into composite systems that ideally build on the strengths of their component algorithms. Collaborative filtering, however, has remained an effective approach, both alone and hybridized with content-based approaches.

Research on recommender algorithms garnered significant attention in 2006 when Netflix launched the Netflix Prize to improve the state of movie recommendation. The objective of this competition was to build a recommender algorithm that could beat their internal CineMatch algorithm in offline tests by 10%. It sparked a flurry of activity, both in academia and amongst hobbyists. The \$1 M prize demonstrates the value that vendors place on accurate recommendations.

## 1.2 Core Concepts, Vocabulary, and Notation

Collaborative filtering techniques depend on several concepts to describe the problem domain and the particular requirements placed

on the system. Many of these concepts are also shared by other recommendation methods.

The information domain for a collaborative filtering system consists of *users* which have expressed preferences for various *items*. A preference expressed by a user for an item is called a *rating* and is frequently represented as a  $(User, Item, Rating)$  triple. These ratings can take many forms, depending on the system in question. Some systems use real- or integer-valued rating scales such as 0–5 stars, while others use binary or ternary (like/dislike) scales.<sup>1</sup> Unary ratings, such as “has purchased”, are particularly common in e-commerce deployments as they express well the user’s purchasing history absent ratings data. When discussing unary ratings, we will use “purchased” to mean that an item is in the user’s history, even for non-commerce settings such as web page views.

The set of all rating triples forms a sparse matrix referred to as the *ratings matrix*.  $(User, Item)$  pairs where the user has not expressed a preference for (rated) the item are unknown values in this matrix. Figure 1.1 shows an example ratings matrix for three users and four movies in a movie recommender system; cells marked ‘?’ indicate unknown values (the user has not rated that movie).

In describing use and evaluation of recommender systems, including collaborative filtering systems, we typically focus on two tasks. The first is the *predict* task: given a user and an item, what is the user’s likely preference for the item? If the ratings matrix is viewed as a sampling of values from a complete user–item preference matrix, than the predict task for a recommender is equivalent to the matrix missing-values problem.

|        | <i>Batman Begins</i> | <i>Alice in Wonderland</i> | <i>Dumb and Dumber</i> | <i>Equilibrium</i> |
|--------|----------------------|----------------------------|------------------------|--------------------|
| User A | 4                    | ?                          | 3                      | 5                  |
| User B | ?                    | 5                          | 4                      | ?                  |
| User C | 5                    | 4                          | 2                      | ?                  |

Fig. 1.1 Sample ratings matrix (on a 5-star scale).

---

<sup>1</sup>The scale is ternary if “seen but no expressed preference” is considered distinct from “unseen”.

The second task is the *recommend* task: given a user, produce the best ranked list of  $n$  items for the user's need. An  $n$ -item recommendation list is not guaranteed to contain the  $n$  items with the highest predicted preferences, as predicted preference may not be the only criteria used to produce the recommendation list.

In this survey, we use a consistent mathematical notation for referencing various elements of the recommender system model. The universe consists of a set  $U$  of users and a set  $I$  of items.  $I_u$  is the set of items rated or purchased by user  $u$ , and  $U_i$  is the set of users who have rated or purchased  $i$ . The rating matrix is denoted by  $\mathbf{R}$ , with  $r_{u,i}$  being the rating user  $u$  provided for item  $i$ ,  $\mathbf{r}_u$  being the vector of all ratings provided by user  $u$ , and  $\mathbf{r}_i$  being the vector of all ratings provided for item  $i$  (the distinction will be apparent from context).  $\bar{r}_u$  and  $\bar{r}_i$  are the average of a user  $u$  or an item  $i$ 's ratings, respectively. A user  $u$ 's preference for an item  $i$ , of which the rating is assumed to be a reflection, is  $\pi_{u,i}$  (elements of the user-item preference matrix  $\mathbf{\Pi}$ ). It is assumed that  $r_{u,i} \approx \pi_{u,i}$ ; specifically,  $\mathbf{R}$  is expected to be a sparse sample of  $\mathbf{\Pi}$  with the possible addition of noise. The recommender's prediction of  $\pi_{u,i}$  is denoted by  $p_{u,i}$ .

### 1.3 Overview

This survey aims to provide a broad overview of the current state of collaborative filtering research. In the next two sections, we discuss the core algorithms for collaborative filtering and traditional means of measuring their performance against user rating data sets. We will then move on to discuss building reliable, accurate data sets; understanding recommender systems in the broader context of user information needs and task support; and the interaction between users and recommender systems.

# 2

---

## Collaborative Filtering Methods

---

*Collaborative filtering* (CF) is a popular recommendation algorithm that bases its predictions and recommendations on the ratings or behavior of other users in the system. The fundamental assumption behind this method is that other users' opinions can be selected and aggregated in such a way as to provide a reasonable prediction of the active user's preference. Intuitively, they assume that, if users agree about the quality or relevance of some items, then they will likely agree about other items — if a group of users likes the same things as Mary, then Mary is likely to like the things they like which she hasn't yet seen.

There are other methods for performing recommendation, such as finding items similar to the items liked by a user using textual similarity in metadata (*content-based filtering* or CBF). The focus of this survey is on collaborative filtering methods, although content-based filtering will enter our discussion at times when it is relevant to overcoming a particular recommender system difficulty.

The majority of collaborative filtering algorithms in service today, including all algorithms detailed in this section, operate by first generating predictions of the user's preference and then produce their recommendations by ranking candidate items by predicted preferences. Often this prediction is in the same scale as the ratings provided by

users, but occasionally the prediction is on a different scale and is meaningful only for candidate ranking. This strategy is analogous to the common information retrieval method of producing relevance scores for each document in a corpus with respect to a particular query and presenting the top-scored items. Indeed, the recommend task can be viewed as an information retrieval problem in which the domain of items (the corpus) is queried with the user's preference profile.

Therefore, this section is primarily concerned with how various algorithms predict user preference. In later sections we will discuss recommendation strategies that diverge from this structure, but in actual implementation they frequently start with a preference-ranked list of items and adjust the final recommendation list based on additional criteria.

## 2.1 Baseline Predictors

Before discussing true collaborative filtering algorithms, let us first consider some baseline prediction methods. These methods are useful for establishing non-personalized baselines against which personalized algorithms can be compared, as well as for pre-processing and normalizing data for use with more sophisticated algorithms. Baseline algorithms that do not depend on the user's ratings can also be useful for providing predictions for new users. We denote the baseline prediction for user  $u$  and item  $i$  by  $b_{u,i}$ .

The simplest baseline is to predict the average rating over all ratings in the system:  $b_{u,i} = \mu$  (where  $\mu$  is the overall average rating). This can be enhanced somewhat by predicting the average rating by that user or for that item:  $b_{u,i} = \bar{r}_u$  or  $b_{u,i} = \bar{r}_i$ .

Baselines can be further enhanced by combining the user mean with the average deviation from user mean rating for a particular item [58, 79, 110, 115]. Generally, a baseline predictor of the following form can be used:

$$b_{u,i} = \mu + b_u + b_i \quad (2.1)$$

$b_u$  and  $b_i$  are user and item baseline predictors, respectively. They can be defined simply by using average offsets as follows, computing

subsequent effects within the residuals of previous effects [14, 58, 115]:

$$b_u = \frac{1}{|I_u|} \sum_{i \in I_u} (r_{u,i} - \mu) \quad (2.2)$$

$$b_i = \frac{1}{|U_i|} \sum_{u \in U_i} (r_{u,i} - b_u - \mu) \quad (2.3)$$

The baseline can be further regularized, providing a more reasonable estimate of user and item preferences in the face of sparse sampling, with the incorporation of damping terms  $\beta_u$  and  $\beta_i$  [44]<sup>1</sup>:

$$b_u = \frac{1}{|I_u| + \beta_u} \sum_{i \in I_u} (r_{u,i} - \mu) \quad (2.4)$$

$$b_i = \frac{1}{|U_i| + \beta_i} \sum_{u \in U_i} (r_{u,i} - b_u - \mu) \quad (2.5)$$

This adjustment causes the baseline predicted ratings to be closer to global mean when the user or item has few ratings, based on the statistical principle that the more ratings a user has given or an item has received, the more is known about that user or item's true mean rating. Funk [44] found that 25 was a useful value for the damping terms.

Additional baselines can be computed and added, and the baselines can be made more sophisticated to deal with various effects [14, 80, 115]. If an item or user is new and therefore has no ratings, its baseline can be set to 0, effectively assuming that it is an average user or item.

The baseline predictors are not restricted to simple ANOVA-style functions. They can also be learned as more general parameters with gradient descent or other parameter estimation techniques, potentially as a part of learning a larger model [14, 79, 80].

Baseline predictors effectively capture effects of user bias, item popularity, and can be applied to more exotic but increasingly-important factors such as time [80, 115]. If the baseline is subtracted from the ratings matrix to yield a normalized ratings matrix  $\hat{\mathbf{R}}$ , all that remains

---

<sup>1</sup>Funk's formulation involved an additional term in the numerator; since we are computing mean offsets rather than means, that term cancels out.

for collaborative filtering to do is to efficiently capture the interaction effect between users and items. Further, the missing values of  $\hat{\mathbf{R}}$  are 0 rather than unknown, simplifying some computations and allowing the matrix to be handled by standard sparse matrix packages.

## 2.2 User–User Collaborative Filtering

*User–user collaborative filtering*, also known as *k-NN collaborative filtering*, was the first of the automated CF methods. It was first introduced in the GroupLens Usenet article recommender [119]. The Ringo music recommender [137] and the BellCore video recommender [62] also used user–user CF or variants thereof.

User–user CF is a straightforward algorithmic interpretation of the core premise of collaborative filtering: find other users whose past rating behavior is similar to that of the current user and use their ratings on other items to predict what the current user will like. To predict Mary’s preference for an item she has not rated, user–user CF looks for other users who have high agreement with Mary on the items they have both rated. These users’ ratings for the item in question are then weighted by their level of agreement with Mary’s ratings to predict Mary’s preference.

Besides the rating matrix  $\mathbf{R}$ , a user–user CF system requires a similarity function  $s: U \times U \rightarrow \mathbb{R}$  computing the similarity between two users and a method for using similarities and ratings to generate predictions.

### 2.2.1 Computing Predictions

To generate predictions or recommendations for a user  $u$ , user–user CF first uses  $s$  to compute a neighborhood  $N \subseteq U$  of neighbors of  $u$ . Once  $N$  has been computed, the system combines the ratings of users in  $N$  to generate predictions for user  $u$ ’s preference for an item  $i$ . This is typically done by computing the weighted average of the neighboring users’ ratings  $i$  using similarity as the weights:

$$p_{u,i} = \bar{r}_u + \frac{\sum_{u' \in N} s(u, u')(r_{u',i} - \bar{r}_{u'})}{\sum_{u' \in N} |s(u, u')|} \quad (2.6)$$

Subtracting the user's mean rating  $\bar{r}_u$  compensates for differences in users' use of the rating scale (some users will tend to give higher ratings than others). Equation (2.6) can also be extended to normalize user ratings to  $z$ -scores by dividing the offset from mean rating by the standard deviation  $\sigma_u$  of each user's ratings, thereby compensating for users differing in rating spread as well as mean rating [58]:

$$p_{u,i} = \bar{r}_u + \sigma_u \frac{\sum_{u' \in N} s(u, u')(r_{u',i} - \bar{r}_{u'})/\sigma_{u'}}{\sum_{u' \in N} |s(u, u')|} \quad (2.7)$$

Weighted averaging is not the only mechanism that has been used for computing predictions, but it is the most common. Ringo used no weighting, performing an unweighted average over ratings by neighboring users [137]. The BellCore system used a multivariate regression over the users in the neighborhood to generate predictions [62]. Weighted averaging is by far the most common, however, as it is simple and works well in practice. It is also consistent with Social Choice theory [111], grounding it in models of human behavior and an axiomatic, first-principles development of collaborative filtering. Social choice theory deals with the preferences of individuals and of a society as a whole. Given several properties that a recommender should exhibit within the social choice framework, Pennock et al. show that weighted averaging is the only aggregation method which produces consistent results.

There remains the question of how many neighbors to select. In some systems, such as the original GroupLens,  $N = U \setminus \{u\}$  (all users are considered as neighbors); in other systems, neighborhoods are selected for each item based on a similarity threshold or neighborhood size such that  $N_i$  is the  $k$  users most similar to  $u$  who have rated the target item  $i$ . Limiting neighborhood size can result in more accurate predictions, as the neighbors with low correlation introduce more noise than signal into the process [58]. The particular threshold to use is domain- and system-specific, so analysis of a relevant data set is needed to choose the neighborhood size for a particular deployment. In offline analysis of available movie ratings data, Herlocker et al. found  $k = 20$  to be a good value; values in the 20–50 range are a reasonable starting point in many domains. Lathia et al. [86] propose dynamically adapting the neighborhood size used for each user to minimize the error in their

predictions as new data is added to the system. Randomly sampling candidate users for the neighborhood can decrease the time required to find neighbors, at the possible expense of accuracy [62].

### 2.2.2 Computing User Similarity

A critical design decision in implementing user–user CF is the choice of similarity function. Several different similarity functions have been proposed and evaluated in the literature.

**Pearson correlation.** This method computes the statistical correlation (Pearson’s  $r$ ) between two user’s common ratings to determine their similarity. GroupLens and BellCore both used this method [62, 119]. The correlation is computed by the following:

$$s(u, v) = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_u \cap I_v} (r_{v,i} - \bar{r}_v)^2}}$$

Pearson correlation suffers from computing high similarity between users with few ratings in common. This can be alleviated by setting a threshold on the number of co-rated items necessary for full agreement (correlation of 1) and scaling the similarity when the number of co-rated items falls below this threshold [58, 59]. Experiments have shown a threshold value of 50 to be useful in improving prediction accuracy, and the threshold can be applied by multiplying the similarity function by  $\min\{|I_u \cap I_v|/50, 1\}$ .

**Constrained Pearson correlation.** Ringo solicited ratings from its users on a 7-point scale, providing a rating guide that fixed 4 as a neutral (neither like nor dislike) value  $r_z$ . With an absolute reference, it is possible to correlate absolute like/dislike rather than relative deviation (as the standard Pearson  $r$  does). This led Shardanand and Maes [137] to propose the constrained Pearson correlation:

$$s(u, v) = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - r_z)(r_{v,i} - r_z)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{u,i} - r_z)^2} \sqrt{\sum_{i \in I_u \cap I_v} (r_{v,i} - r_z)^2}}$$

**Spearman rank correlation.** The Spearman rank correlation coefficient is another candidate for a similarity function [58]. For the Spearman correlation, the items a user has rated are ranked such that their highest-rated item is at rank 1 and lower-rated items have higher ranks. Items with the same rating are assigned the average rank for their position. The computation is then the same as that of the Pearson correlation, except that ranks are used in place of ratings.

**Cosine similarity.** This model is somewhat different than the previously described approaches, as it is a vector-space approach based on linear algebra rather than a statistical approach. Users are represented as  $|I|$ -dimensional vectors and similarity is measured by the cosine distance between two rating vectors. This can be computed efficiently by taking their dot product and dividing it by the product of their  $L_2$  (Euclidean) norms:

$$s(u, v) = \frac{\mathbf{r}_u \cdot \mathbf{r}_v}{\|\mathbf{r}_u\|_2 \|\mathbf{r}_v\|_2} = \frac{\sum_i r_{u,i} r_{v,i}}{\sqrt{\sum_i r_{u,i}^2} \sqrt{\sum_i r_{v,i}^2}}$$

Unknown ratings are considered to be 0; this causes them to effectively drop out of the numerator. If the user mean baseline is subtracted from the ratings prior to computing the similarity, cosine similarity is equivalent to Pearson correlation when the users have rated the same set of items and decreases as  $\frac{|I_u \cap I_v|^2}{|I_u||I_v|}$  decreases.

The threshold-based damping described for Pearson correlation can, in principle, be applied to any similarity function, although the benefits of doing so have not been studied. An alternative method for damping is to add a large constant, such as 100, to the denominator; this attenuates the similarity of users with small sets of co-rated items.

Other similarity functions, such as mean-squared difference [137], have been proposed, but have not seen significant adoption. In general, Pearson correlation has been found to provide the best results [22, 58], although results from the Ringo system suggest that constrained Pearson correlation may provide some improvement when items are rated on an absolute scale [137].

|        | <i>Batman Begins</i> | <i>Alice in Wonderland</i> | <i>Dumb and Dumber</i> | <i>Equilibrium</i> |
|--------|----------------------|----------------------------|------------------------|--------------------|
| User A | 4                    | ?                          | 3                      | 5                  |
| User B | ?                    | 5                          | 4                      | ?                  |
| User C | 5                    | 4                          | 2                      | ?                  |
| User D | 2                    | 4                          | ?                      | 3                  |
| User E | 3                    | 4                          | 5                      | ?                  |

Fig. 2.1 Example ratings matrix (on a 5-star scale).

### 2.2.3 Example

Consider the ratings matrix in Figure 2.1 (an extension of the sample matrix in Figure 1.1). We want to find User C’s prediction for *Equilibrium* ( $p_{C,e}$ ) with the following configuration:

- Pearson correlation.
- Neighborhood size of 2.
- Weighted average with mean offset (Equation (2.6)).

C’s mean rating is 3.667. There are only two users who have rated *Equilibrium*, and therefore only two candidate users for the neighborhood: A and D.  $s(C,A) = 0.832$  and  $s(C,D) = -0.515$ . The prediction  $p_{C,e}$  is therefore computed as follows:

$$\begin{aligned}
 p_{C,e} &= \bar{r}_C + \frac{s(C,A)(r_{A,e} - \bar{r}_A) + s(C,D)(r_{D,e} - \bar{r}_D)}{|s(C,A)| + |s(C,D)|} \\
 &= 3.667 + \frac{0.832 \cdot (5 - 4) + -0.515 \cdot (2 - 3)}{0.832 + 0.515} \\
 &= 4.667
 \end{aligned}$$

## 2.3 Item–Item Collaborative Filtering

User–user collaborative filtering, while effective, suffers from scalability problems as the user base grows. Searching for the neighbors of a user is an  $O(|U|)$  operation (or worse, depending on how similarities are computing — directly computing most similarity functions against all other users is linear in the total number of ratings). To extend collaborative filtering to large user bases and facilitate deployment on e-commerce sites, it was necessary to develop more scalable

algorithms. *Item-item collaborative filtering*, also called *item-based* collaborative filtering, takes a major step in this direction and is one of the most widely deployed collaborative filtering techniques today.

Item-item collaborative filtering was first described in the literature by Sarwar et al. [130] and Karypis [71], although a version of it seems to have been used by Amazon.com at this time [87]. Rather than using similarities between users' rating behavior to predict preferences, item-item CF uses similarities between the rating patterns of items. If two items tend to have the same users like and dislike them, then they are similar and users are expected to have similar preferences for similar items. In its overall structure, therefore, this method is similar to earlier content-based approaches to recommendation and personalization, but item similarity is deduced from user preference patterns rather than extracted from item data.

In its raw form, item-item CF does not fix anything: it is still necessary to find the most similar items (again solving the  $k$ -NN problem) to generate predictions and recommendations. In a system that has more users than items, it allows the neighborhood-finding to be amongst the smaller of the two dimensions, but this is a small gain. It provides major performance gains by lending itself well to pre-computing the similarity matrix.

As a user rates and re-rates items, their rating vector will change along with their similarity to other users. Finding similar users in advance is therefore complicated: a user's neighborhood is determined not only by their ratings but also by the ratings of other users, so their neighborhood can change as a result of new ratings supplied by any user in the system. For this reason, most user-user CF systems find neighborhoods at the time when predictions or recommendations are needed.

In systems with a sufficiently high user to item ratio, however, one user adding or changing ratings is unlikely to significantly change the similarity between two items, particularly when the items have many ratings. Therefore, it is reasonable to pre-compute similarities between items in an item-item similarity matrix. The rows of this matrix can even be truncated to only store the  $k$  most similar items. As users change ratings, this data will become slightly stale, but the users will

likely still receive good recommendations and the data can be fully updated by re-computing the similarities during a low-load time for the system.

Item–item CF generates predictions by using the user’s own ratings for other items combined with those items’ similarities to the target item, rather than other users’ ratings and user similarities as in user–user CF. Similar to user–user CF, the recommender system needs a similarity function, this time  $s: I \times I \rightarrow \mathbb{R}$ , and a method to generate predictions from ratings and similarities.

### 2.3.1 Computing Predictions

In real-valued ratings domains, the similarity scores can be used to generate predictions using a weighted average, similar to the procedure used in user–user CF. Recommendations are then generated by picking the candidate items with the highest predictions.

After collecting a set  $S$  of items similar to  $i$ ,  $p_{u,i}$  can be predicted as follows [130]:

$$p_{u,i} = \frac{\sum_{j \in S} s(i,j) r_{u,j}}{\sum_{j \in S} |s(i,j)|} \quad (2.8)$$

$S$  is typically selected to be the  $k$  items most similar to  $j$  that  $u$  has also rated for some neighborhood size  $k$ . Sarwar et al. [130] found  $k = 30$  produced good results on the MovieLens data set.

Equation (2.8) as it stands suffers from two deficiencies. The first comes to light when it is possible for similarity scores to be negative and ratings are constrained to be nonnegative: some of the ratings averaged together to compute the prediction may be negative after weightings. While this will not affect the relative ordering of items by predicted value, it will bias the predicted values so they no longer map back to the user rating domain. This can be corrected either by thresholding similarities so only items with nonnegative similarities are considered or by averaging distance from the baseline predictor:

$$p_{u,i} = \frac{\sum_{j \in S} s(i,j)(r_{u,j} - b_{u,i})}{\sum_{j \in S} |s(i,j)|} + b_{u,i} \quad (2.9)$$

The other difficulty is with non-real-valued ratings scales, particularly the unary scales common on e-commerce sites without ratings data. In this case, the averaging does not work: if all similarities are positive and  $r_{u,i} = 1$  if user  $u$  has purchased item  $i$ , then Equation (2.8) always evaluates to 1. With negative similarities, it is similarly ill-behaved. To work around this, we can compute pseudo-predictions  $\tilde{p}_{u,i}$  with a simple aggregation of the similarities to items in the user's purchase history  $I_u$ . Summation has been tested and shown to perform well (Equation (2.10)); other aggregations such as mean or max may also be considered or used in practice.

$$\tilde{p}_{u,i} = \sum_{j \in I_u} s(i,j) \quad (2.10)$$

$\tilde{p}_{u,i}$  is not in a meaningful scale to predict any particular user behavior, but the predict task is typically not as important in unary contexts. This pseudo-prediction can, however, be used to rank candidate items for recommendation, forming a good basis for using item-item CF to recommend items based on user purchase histories [38, 71, 87].

It is also possible to use weights other than the similarity function for computing the final prediction. Bell and Koren [14] proposed a formulation that computes item weights directly by estimating, for each user-item pair  $u,i$ , the solution to the linear equation  $\mathbf{Aw} = \mathbf{b}$ . The solution  $\mathbf{w}$  is such that  $w_j$  is the optimal weight to use for  $u$ 's rating of  $j$  in computing their rating of  $i$ , and  $\mathbf{A}$  and  $\mathbf{b}$  are given as follows:

$$a_{j,k} = \sum_{v \neq u} \pi_{v,j} \pi_{v,k} \quad (2.11)$$

$$b_j = \sum_{v \neq u} \pi_{v,j} \pi_{v,i} \quad (2.12)$$

The computed weights, differing for each user-item pair, are then used to compute the prediction  $p_{u,i} = \sum_{j \in S} w_j r_{u,j}$ .

### 2.3.2 Computing Item Similarity

The item-item prediction process requires an item-item similarity matrix  $\mathbf{S}$ . This matrix is a standard sparse matrix, with missing values

being 0 (no similarity); it differs in this respect from  $\mathbf{R}$ , where missing values are unknown.

As in user–user collaborative filtering, there are a variety of methods that can be used for computing item similarities.

**Cosine similarity.** Cosine similarity between item rating vectors is the most popular similarity metric, as it is simple, fast, and produces good predictive accuracy.

$$s(i, j) = \frac{\mathbf{r}_i \cdot \mathbf{r}_j}{\|\mathbf{r}_i\|_2 \|\mathbf{r}_j\|_2}$$

**Conditional probability.** For domains with unary ratings (such as shopping site purchase histories), Karypis [71] proposed a similarity function based on conditional probabilities:  $s(i, j) = \Pr_B[j \in B | i \in B]$  where  $B$  is a user’s purchase history. With a scaling parameter  $\alpha$  to balance for frequently occurring items, this formula becomes

$$s(i, j) = \frac{\text{Freq}(i \wedge j)}{\text{Freq}(i)(\text{Freq}(j))^\alpha}$$

$\alpha$  serves as a damping factor to compensate for  $\Pr_B[j \in B | i \in B]$  being high solely due to a high marginal probability  $\Pr_B[j \in B]$  (if  $j$  is an item that many people purchase, it will be similar to most items;  $\alpha$  reduces this effect to bring items that are more uniquely similar to the top). This normalization hinders the ability to generate true predictions, but this is not a problem in unary domains due to the use of pseudo-predictions.

Notably, this function is not symmetric ( $s(i_1, i_2)$  may not be the same as  $s(i_2, i_1)$ ). Care must be taken that it is used in the correct orientation.

**Pearson correlation.** Pearson correlation has also been proposed for item–item recommendation, but does not seem to work as well as cosine similarity [130].

In order to optimize the recommender’s performance, it is important to normalize the ratings prior to computing the similarity matrix [14, 130]. In real-valued domains variation in ratings due to user ratings

bias (e.g., two users liking and disliking similar films, but one being a cynic who rates average films 2.5/5 and the other an enthusiast who rates the average film at 4/5) and item bias allows the collaborative filter to focus on the more nuanced differences in user preference for particular items. This can be accomplished by subtracting a baseline predictor from all ratings prior to computing similarities (e.g., compute similarities over ratings  $\hat{r}_{u,i} = r_{u,i} - \mu - b_u - b_i$ ).

When applying cosine similarity in unary ratings domains, it can be useful to normalize each user's ratings vector  $\mathbf{r}_u$  to the unit vector prior to computing item similarities. The effect of this adjustment is that users who have purchased fewer items have more impact on the similarity of the items they have purchased than users who have purchased many items [38, 71].

Similarly, normalizing item similarity vectors (rows of  $\mathbf{S}$ ) to unit vectors can be beneficial. This causes items with sparser neighborhoods (fewer similar items) to have more influence in computing the final predictions [38, 71].

### 2.3.3 Pre-computing and Truncating the Model

Due to the relatively static nature of item similarities when  $|U| \gg |I|$  (particularly when there are more ratings-per-item than ratings-per-user), it is feasible to pre-compute item-item similarities and cache the  $k'$  most similar items to each item. Prediction can then be performed quickly by looking up the similarity list for each item rated by the current user and aggregating their similarities into a predicted preference. Caching more items than are used in the similarity computation (so  $k' > k$ ) is useful to increase the likelihood of having  $k$  similar items after items already rated by the user have been removed from the candidate set.

Pre-computation and truncation is essential to deploying collaborative filtering in practice, as it places an upper bound on the number of items which must be considered to produce a recommendation and eliminates the query-time cost of similarity computation. It comes with the small expense of reducing the number of items for which predictions can be generated (the *coverage* of the recommender), but the

unrecommendable items will usually have low predicted preferences anyway.

### 2.3.4 Example

Again using the example ratings in Figure 2.1, let us now compute C's prediction for *Equilibrium* using item–item CF with cosine similarity. We compute the length ( $L_2$  norm) of each movie's vector, its norm with the vector  $r_e$  for *Equilibrium*, and finally the cosine similarity:

| Movie                          | $\ \mathbf{r}\ _2$ | $\mathbf{r} \cdot \mathbf{r}_e$ | $\frac{\mathbf{r} \cdot \mathbf{r}_e}{\ \mathbf{r}\ _2 \ \mathbf{r}_e\ _2}$ |
|--------------------------------|--------------------|---------------------------------|---|
| <i>Batman Begins</i> (b)       | 7.348              | 24                              | 0.607   |
| <i>Alice in Wonderland</i> (a) | 8.544              | 8                               | 0.174   |
| <i>Dumb and Dumber</i> (d)     | 7.348              | 15                              | 0.382   |
| <i>Equilibrium</i> (e)         | 5.385              |                                 |   |

User C has rated all three other movies, but we will only use the most similar two when computing the prediction:

$$\begin{aligned} p_{C,e} &= \frac{s(b,e) \cdot r_{C,b} + s(d,e) \cdot r_{C,d}}{|s(b,e)| + |s(d,e)|} \\ &= \frac{0.607 * 5 + 0.382 * 2}{0.607 + 0.382} \\ &= 3.84 \end{aligned}$$

## 2.4 Dimensionality Reduction

In both of the traditional collaborative filtering algorithms so far described, there are hints of viewing the user–item ratings domain as a vector space. With this view, however, the vectors are of extremely high dimension: an item is a  $|U|$ -dimensional vector with missing values of users' preferences for it (similarly, a user is a  $|I|$ -dimensional vector). Further, there is redundancy in these dimensions, as both users and items will usually be divisible into groups with similar preference profiles (e.g., many science fiction movies will be liked to similar degrees by the same set of users). It is therefore natural to ask whether the dimensionality of the rating space can be reduced — can we find a smaller number of dimensions, ideally a constant number  $k$ , so that items and users can be represented by  $k$ -dimensional vectors?

In particular, we want to identify a set of  $k$  *topics* so that user preferences for items can be expressed as a combination of the user's interest in a topic (*topic interest*) and the extent to which each item is relevant to the topic (*topic relevance*). Some recommendation methods do this explicitly using content features, tags attached to the items, or user clustering or stereotyping. Latent semantic analysis, a technique pioneered in information retrieval, provides a way to do this decomposition using only the rating data. The topics are considered to be latent in the rating data.

In information retrieval, a document corpus can be represented as a term-document matrix where each cell is the number of times the given term occurs in a particular document. This results in high-dimensional representations of terms and documents, further complicated by the problems of synonymy (different terms having the same or similar meaning), polysemy (the same term having different meanings), and noise (documents or queries using terms incorrectly). Latent semantic analysis (LSA, also called latent semantic indexing or LSI) deals with these problems by using dimensionality reduction, in the form of truncated singular value decomposition (SVD), to extract the semantic relationships between documents latent in their use of vocabulary [16, 36]. SVD-based dimensionality reduction has since been adapted to collaborative filtering by Billsus and Pazzani [18], Sarwar et al. [128, 131], and many others.

#### 2.4.1 Defining Singular Value Decomposition

For a matrix  $\mathbf{M}$ , its SVD is the factorization of  $\mathbf{M}$  into three constituent matrices such that  $\mathbf{M} = \mathbf{U}\Sigma\mathbf{T}^T$ ,  $\Sigma$  is a diagonal matrix whose values  $\sigma_i$  are the *singular values* of the decomposition, and both  $\mathbf{U}$  and  $\mathbf{T}$  are orthogonal. What this accomplishes is introducing an intermediate vector space represented by  $\Sigma$ . If  $M$  is the ratings matrix,  $\Sigma\mathbf{T}^T$  transforms vectors from item-space into the intermediate vector space.

In the pure form of the SVD,  $\mathbf{U}$  is  $m \times \hat{k}$ ,  $\Sigma$  is  $k \times \hat{k}$ , and  $\mathbf{V}$  is  $n \times \hat{k}$ , where  $\mathbf{M}$  is  $m \times n$  and has rank  $\hat{k}$ ; this is not a significant gain.  $\Sigma$  can, however, be truncated by only retaining the  $k$  largest singular values to yield  $\Sigma_k$ . The resulting decomposition is an approximation

of  $M$ . Further, using the Frobenius norm as the measure of error, it is the best possible rank- $k$  approximation [36].

This truncation simultaneously achieves two goals. First, it decreases the dimensionality of the vector space, decreasing the storage and computational requirements for the model. Items and users can each be represented by  $k$ -dimensional vectors. Second, by dropping the smaller singular values, small perturbances as a result of noise in the data are eliminated, leaving only the strongest effects or trends in the model. In collaborative filtering, this noise can come as a result of other factors besides sheer preference playing a role in a user's rating; decreasing the impact of noise improves our ability to provide high-quality recommendations.

Computing the SVD of the ratings matrix results in the following factorization, with  $m = |U|$ ,  $n = |I|$ , and  $\Sigma$  a  $k \times k$  diagonal matrix (also shown in Figure 2.2):

$$\mathbf{R} \approx \mathbf{U}\Sigma\mathbf{T}^T \quad (2.13)$$

Once the rank- $k$  SVD of Equation (2.13) has been computed, it can be interpreted as an expression of the topic preference-relevance model. The rows of the  $|U| \times k$  matrix  $\mathbf{U}$  are the users' interest in each of the  $k$  inferred topics, and the rows of  $\mathbf{I}$  are the item's relevance for each topic. The singular values in  $\Sigma$  are weights for the preferences, representing the influence of a particular topic on user-item preferences across the system. A user's preference for an item, therefore, is the weighted sum of the user's interest in each of the topics times that item's relevance to the topic.

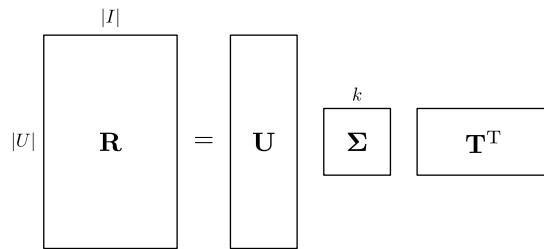


Fig. 2.2 Singular value decomposition of the ratings matrix.

### 2.4.2 Computing and Updating the SVD

In order to use SVD (or any matrix factorization method), it is necessary to first compute the matrix factorization. There are a variety of algorithms for computing singular value decompositions, including Lanczos' algorithm, the generalized Hebbian algorithm, and expectation maximization [51, 81, 127].

Singular value decomposition is only well-defined when the matrix is complete. Therefore, to factor the ratings matrix, the missing values must be filled with some reasonable default (a method called *imputation*). Sarwar et al. [131] found the item's average rating to be a useful default value (they tried user average as well, but item average performed better). Alternatively, the SVD can be computed over the normalized ratings matrix  $\hat{\mathbf{R}}$  (see Section 2.1) and the missing values considered to be 0.

Several methods have been proposed that compute an estimate of the SVD only on the known ratings, dropping the requirement to impute or otherwise account for missing ratings. Kurucz et al. [81] propose a least-squares method that learns a regression for each user. Another method that has become quite popular in the last few years is gradient descent [44, 110]. This method trains each topic  $f$  in turn, using the following update rules ( $\lambda$  is the learning rate, typically 0.001):

$$\Delta u_{j,f} = \lambda(r_{u,i} - p_{u,i})i_{k,f} \quad (2.14)$$

$$\Delta i_{k,f} = \lambda(r_{u,i} - p_{u,i})u_{j,f} \quad (2.15)$$

The gradient descent method for estimating the SVD also allows for regularization to prevent overfitting the resulting model. The resulting model will not be a true SVD of the rating matrix, as the component matrices are no longer orthogonal, but tends to be more accurate at predicting unseen preferences than the unregularized SVD. The regularization is accomplished by adding an additional term to the update rules in Equations (2.14) and (2.15) [44].  $\gamma$  is the regularization factor, typically 0.1–0.2.

$$\Delta u_{j,f} = \lambda((r_{u,i} - p_{u,i})i_{k,f} - \gamma u_{j,f}) \quad (2.16)$$

$$\Delta i_{k,f} = \lambda((r_{u,i} - p_{u,i})u_{j,f} - \gamma i_{k,f}) \quad (2.17)$$

Prior to computing the SVD, the ratings can additionally be normalized, e.g., by subtracting the user's mean rating or some other baseline predictor. This can improve both accuracy [131] and accelerate convergence of iterative methods [44].

Once the SVD is computed, it is necessary to update it to reflect new users, items, and ratings. A commonly used method for updating the SVD is *folding-in*; it works well in practice and allows users who were not considered when the ratings matrix was factored to receive recommendations and predictions [16, 129]. Folding-in operates by computing a new user-preference or topic-relevance vector for the new user or item but not recomputing the decomposition itself.

For a user  $u$ , folding-in computes their topic interest vector  $\mathbf{u}$  such that  $\pi_u \approx \mathbf{u}\Sigma\mathbf{T}^T$ . Given their rating vector  $\mathbf{r}_u$ ,  $\mathbf{u}$  can therefore be calculated as  $\mathbf{u} = (\Sigma\mathbf{T}^T)^{-1}\mathbf{r}_u = \mathbf{T}\Sigma^{-1}\mathbf{r}_u$ . Setting unknown ratings to 0 causes the folding-in process to ignore them, and a new user preference vector is now available. The same process works for item vectors, except  $\mathbf{U}$  is substituted for  $\mathbf{T}$ .

As user and item vectors are updated with the folding-in process, the accuracy of the SVD will diminish over time. It is therefore necessary to periodically re-compute the complete factorization. In a deployed system, this can be done off-line during low-load times [129].

Brand [21] proposed an alternative method for building and maintaining the SVD based on rank-1 updates. His method produces fast, real-time updates of the SVD, bootstrapping the SVD with a dense portion of the data set. This dense portion can be extracted by sorting users and items to make a dense corner in the ratings matrix.

### 2.4.3 Generating Predictions

The predicted preference of user  $u$  for item  $i$  can be computed as the weighted dot product of the user-topic interest vector  $\mathbf{u}$  and the item-topic relevance vector  $\mathbf{i}$ :

$$p_{u,i} = \sum_f u_f \sigma_f i_f \quad (2.18)$$

Recommendation can be done by ranking items in order of predicted preference. The user's preference  $\mathbf{p}_u$  for all items can be computed

efficiently by multiplying their topic-interest vector with the scaled item matrix  $\Sigma\mathbf{T}^T$ :

$$\mathbf{p}_u = \mathbf{u}\Sigma\mathbf{T}^T \quad (2.19)$$

Recommendations and predictions for users who were not considered in the factorization can be computed by applying the folding-in process to compute a user preference vector given a rating vector.

#### 2.4.4 Computing Similarities

Singular value decomposition can be used to generate user and item neighborhoods as well as predictions. User–user similarity can be computed using a vector similarity metric between the two users’ topic interest vectors (rows in  $\mathbf{U}$ ). Since  $\mathbf{U}$  is orthogonal, a dot product between two rows suffices to compute their cosine similarity. Item similarities can be found similarly by operating on rows of  $\mathbf{T}$  [36, 131].

These similarities can be used to compute neighborhoods which in turn produce recommendations [50, 128, 131]. Using neighborhoods to generate recommendations can be particularly useful in e-commerce domains with unary rating sets, as recommendations can be computed by summing the similarities of items to each of the items in the user’s shopping cart.

#### 2.4.5 Normalization in SVD

As with item–item CF, it is beneficial to normalize ratings by subtracting baseline predictors prior to computing the model [44, 110]. When doing this, however, it is necessary to also use the baseline when computing predictions. In item–item CF, since the similarity computations are only used as weights applied to averaging the user’s other ratings, normalization in the prediction step is less important; with matrix factorization, the normalization is encoded into the decomposed matrix and therefore must be reversed at the prediction stage. If the baseline predictor  $b_{u,i}$  has been subtracted from ratings prior to model computation, the prediction rule then becomes

$$p_{u,i} = b_{u,i} + \sum_f u_f \sigma_f i_f$$

The baseline predictor can be incorporated into the model learning process when using a gradient descent method, thus allowing for the recommender to learn user and item biases that may differ somewhat from plain means [79, 110].

#### 2.4.6 Principle Component Analysis

Singular value decomposition is not the only matrix factorization method used for collaborative filtering. The Eigentaste algorithm used in Jester uses *principle component analysis* (PCA) on the dense matrix of user ratings for the “gauge set” of jokes rated by all users [50]. Eigentaste normalizes the subset of  $\mathbf{R}$  consisting of the ratings of gauge set jokes by the mean and standard deviation of the ratings of each joke, resulting in a normalized ratings matrix  $\hat{\mathbf{R}}$  ( $\hat{r}_{u,i} = (r_{u,i} - \mu_i)/\sigma_i$ , where  $\mu_i$  is the mean rating for item  $i$  and  $\sigma_i$  the standard deviation of ratings of  $i$ ). PCA then computes the correlation matrix  $\mathbf{C} = \frac{1}{|U|-1}\hat{\mathbf{R}}^T\hat{\mathbf{R}}$  and a factorization  $\mathbf{C} = \mathbf{E}^T\Lambda\mathbf{E}$ , where  $\Lambda$  is a diagonal matrix with  $\lambda_i$  being the  $i$ th eigenvalue of  $\mathbf{C}$ . The top  $k$  eigenvalues are retained, and the resulting factorization projects users into  $k$ -dimensional space. Eigentaste then clusters users in the  $k$ -dimensional space (with  $k = 2$ ) by recursive rectangular clustering and recommends jokes based on the preferences of other users in a user’s cluster.

### 2.5 Probabilistic Methods

Besides the probabilistic item similarity functions discussed in *Computing Item Similarity*, several fully probabilistic formulations of collaborative filtering have been proposed and gained some currency. These methods generally aim to build probabilistic models of user behavior and use those models to predict future behavior. The core idea of probabilistic methods is to compute either  $P(i|u)$ , the probability that user  $u$  will purchase or view item  $i$ , or the probability distribution  $\mathbf{P}(r_{u,i}|u)$  over user  $u$ ’s rating of item  $i$  (and the related problem  $E[r_{u,i}]$ , the expected value of  $u$ ’s rating of  $i$ ).

Cross-sell [73] uses pairwise conditional probabilities with the naïve Bayes assumption to do recommendation in unary e-commerce domains. Based on user purchase histories, the algorithm estimates

$P(a|b)$  (the probability that a user purchases  $a$  given that they have purchased  $b$ ) for each pair of items  $a, b$ . The user's currently-viewed item or shopping basket is combined with these pairwise probabilities to recommend items optimizing the expected value of site-defined objective functions (such as profit or sales lift).

Personality diagnosis [112] is a probabilistic user model that assumes that a user's ratings are a combination of their preference and Gaussian noise ( $r_{u,i} \sim N(\pi_{u,i}, \sigma)$ , where  $\sigma$  is a parameter to the recommendation model). For a user  $u$ , the probability of their true preference profile being equal to that of each other user  $u'$  ( $P(\pi_u = \mathbf{r}_{u'})$ ) is calculated and used to compute a probability distribution over ratings for items  $u$  has not yet rated. Prediction and recommendation are both done by computing the expected value of the user's rating using the resulting distribution.

### 2.5.1 Probabilistic Matrix Factorization

Probabilistic latent semantic analysis (PLSA, also called PLSI or probabilistic latent semantic indexing in the information retrieval literature) is a matrix factorization technique similar to singular value decomposition but arising from statistical probability theory rather than linear algebra [64, 65]. Jin et al. [67] applied it to mining web usage patterns; this led to its later application to recommender systems as a means to protect collaborative filtering results from manipulation by users Mobasher et al. [103].

The basis of PLSA is a probabilistic mixture model of user behavior, diagrammed with plate notation in Figure 2.3(a). PLSA decomposes the probability  $P(i|u)$  by introducing a set  $Z$  of latent factors. It assumes that the user selects an item to view or purchase by selecting a factor  $z \in Z$  with probability  $P(z|u)$  and then selecting an item with probability  $P(i|z)$ ;  $P(i|u)$  is therefore  $\sum_z P(i|z)P(z|u)$ . This has the effect of representing users as a *mixture* of preference profiles or feature preferences (represented as a probability distribution over factors), and attributing item preference to the preference profiles rather than directly to the users. The probabilities can be learned using approximation methods for Bayesian inference such as expectation

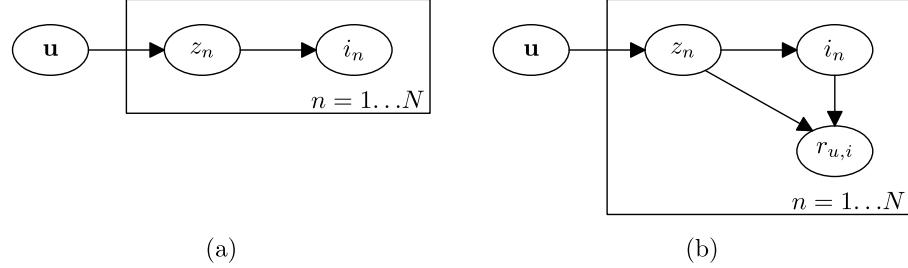


Fig. 2.3 PLSI generative model: (a) models user purchases and (b) models real-valued ratings varying by item.

maximization [37]. The probabilities can be stored in matrices, so that the preference matrix  $\mathbf{P}$  (where  $p_{u,i} = P(i|u)$ ) is decomposed into

$$\mathbf{P} = \hat{\mathbf{U}}\Sigma\hat{\mathbf{T}}^T$$

$\hat{\mathbf{U}}$  is the matrix of the mixtures of preference profiles for each user (so  $\hat{u}_{u,z} = P(z|u)$ ) and  $\hat{\mathbf{T}}$  is the matrix of preference profile probabilities of selecting various items.  $\Sigma$  is a diagonal matrix such that  $\sigma_z = P(z)$  (the marginal probability or global weight of factor  $z$ ). This factorization allows prediction to be done meaningfully in unary domains by considering the probability that  $u$  will purchase  $i$ , in contrast to item-item unary recommendation where the psuedo-predictions were only useful for ranking candidate items.

Hofmann [64] discusses in more detail the particular relationship between PLSA and SVD. The end result — a factorization of the ratings matrix into three component matrices — is quite similar, but the SVD is determined by minimizing error while PLSA is computed by maximizing the predictive power of the model. In the SVD,  $\mathbf{U}$  and  $\mathbf{T}$  are orthogonal matrices leading to a clear geometric or vector-space interpretation of the model, while the PLSA component matrices  $\hat{\mathbf{U}}$  and  $\hat{\mathbf{T}}$  are stochastic matrices with a natural probabilistic interpretation mapping more directly to generative models of user behavior.

PLSA can be extended to model ratings as well as purchases by introducing the rating  $r_{u,i}$  as a new random variable and assuming that it is dependent on the latent topic  $z$  and either the specific user or item under consideration, as shown in Figure 2.3(b) [65]. If the user

is used, then it is assumed that  $u$ 's preference for  $i$  is determined only by their preference for the latent factors  $z$  generating  $i$ ; if the item is used, then all users with the same preference for a factor  $z$  will have the same preference for any item generated by that factor, with variation dependent only on the individual items. The resulting networks can again be learned by expectation maximization.

Blei et al. [20] extended the PLSI aspect model by using a Dirichlet prior over user preferences for topics. This method is called *latent Dirichlet allocation* and yields a more thorough generative model that accounts not only for users purchasing items but also for users coming into the system's knowledge. Users, represented by their preference for factors (the distribution  $P(z|u)$ ), are considered to be instances of a random variable drawn from a Dirichlet distribution. This model, depicted in Figure 2.4, requires two parameters to be learned. For a model with  $k$  factors, these are  $\alpha$ , a  $k$ -dimensional vector parameterizing the Dirichlet distribution from which users are drawn, and  $\beta$ , the  $k \times |I|$  topic-item probability matrix (representing  $P(i|z)$ ).

Another probabilistic method related to matrix factorization is *factor analysis* [27]. This method learns a probabilistic variant of a linear regression  $\mathbf{R}^T = \Lambda \mathbf{U} + \mathbf{N}$ , where  $\Lambda$  is an  $|I| \times k$  model matrix;  $\mathbf{U}$ , a  $k \times |U|$  matrix of user preferences for each of  $k$  different factors, and  $\mathbf{N}$ , a Gaussian noise matrix. By using expectation maximization to learn  $\Lambda$  and the variance of the distribution from which the elements of  $\mathbf{N}$  are drawn, a model can be computed for deducing user preferences for factors from ratings and using those preferences to produce recommendations.

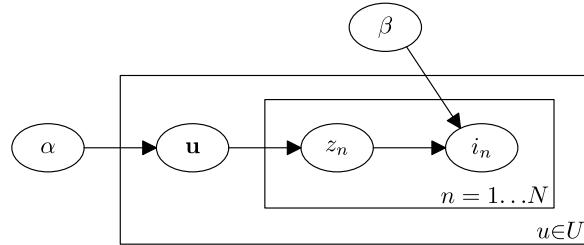


Fig. 2.4 LDA model of users and purchases.

### 2.5.2 Machine Learning Methods

A variety of other methods, usually probabilistic, from the machine learning and artificial intelligence literature have also been applied to recommendation. Some formulations of recommendation can be studied as classifier problems [22]. Recommenders have been built using Bayesian networks [30, 157], Markov decision processes [136], and neural networks [125].

## 2.6 Hybrid Recommenders

It is natural to consider combining several different recommender algorithms into a hybrid recommender system [24]. In some applications, hybrids of various types have been found to outperform individual algorithms [146]. Hybrids can be particularly beneficial when the algorithms involved cover different use cases or different aspects of the data set. For example, item-item collaborative filtering suffers when no one has rated an item yet, but content-based approaches do not. A hybrid recommender could use description text similarity to match the new item with existing items based on metadata, allowing it to be recommended anyway, and increase the influence of collaborative filtering as users rate the item; similarly, users can be defined by the content of the items they like as well as the items themselves. Fab used such an approach, matching items against both the content of items liked by the user and the content of items liked by similar users [13].

Burke [24] provides a thorough analysis of hybrid recommender systems, grouping them into seven classes:

- *Weighted* recommenders take the scores produced by several recommenders and combine them to generate a recommendation list (or prediction) for the user.
- *Switching* recommenders switch between different algorithms and use the algorithm expected to have the best result in a particular context.
- *Mixed* recommenders present the results of several recommenders together. This is similar to weighting, but the results are not necessarily combined into a single list.

- *Feature-combining* recommenders use multiple recommendation data sources as inputs to a single meta-recommender algorithm.
- *Cascading* recommenders chain the output of one algorithm into the input of another.
- *Feature-augmenting* recommenders use the output of one algorithm as one of the input features for another.
- *Meta-level* recommenders train a model using one algorithm and use that model as input to another algorithm.

Hybrid recommenders proved to be quite powerful in the Netflix Prize: the winning entry was a hybrid of some 100 separate algorithms. One important development to come from the prize competition is *feature-weighted linear stacking*, a new weighting hybrid method used by some of the top-ranking teams. Many weighting algorithms use a linear combination of the predictions of many recommenders:

$$p_{u,i} = \alpha_1 p_{u,i}^{(1)} + \cdots + \alpha_n p_{u,i}^{(n)} \quad (2.20)$$

Feature-weighted linear stacking replaces each coefficient  $\alpha_j$  with a function  $g_j$  of item meta-features, such as number of ratings or genre, to alter the blending ratio of the various algorithms' predictions on an item-by-item basis:

$$p_{u,i} = g_1(i) p_{u,i}^{(1)} + \cdots + g_n(i) p_{u,i}^{(n)} \quad (2.21)$$

This allows, for example, the relative weights of item–item CF and actor list similarity to shift based on the number of ratings an item has received.

## 2.7 Selecting an Algorithm

The various algorithms presented each have their strengths and weaknesses. User–user and item–item CF algorithms are well-understood, easy to implement, and provide reasonable performance in most cases. User-based algorithms tend to be more tractable when there are more items than users, and item-based when the situation is reversed. Both methods also require minimal offline computation at the expense of

somewhat greater computational demands when recommendations are needed (precomputing item-item CF improves query-time performance at the expense of more offline computation). User-user CF seems to provide greater serendipity in its recommendations; in the case of MovieLens, this resulted in greater user satisfaction.

Matrix factorization methods require expensive offline model computation steps. The resulting models, however, are very fast for online use (prediction is simply a dot product). They are also capable of reducing the impact of ratings noise and can hamper the ability of users to manipulate the output of the system by decreasing their direct impact on each others' ratings. Probabilistic models are applicable when the recommendation process should follow models of user behavior.

In the end, the appropriate algorithm for a particular application should be selected based on a combination of the characteristics of the target domain and context of use, the computational performance requirements of the application, and the needs of the users of the system. *User Information Needs* discusses various needs users may have from the recommender system. Understanding more fully what needs are best met by what algorithms is a subject of ongoing research.

# 3

---

## Evaluating Recommender Systems

---

When developing a recommender system, either a new algorithm or a new application, it is useful to be able to evaluate how well the system works. Since recommendation is usually a means to some other goal (user satisfaction, increased sales, etc.), testing ultimately needs to take this into account and measure the intended effect. However, it can be costly to try algorithms on real sets of users and measure the effects. Further, measuring some desired effects may be intractable or impossible, resulting in the need for plausible proxies.

In the recommender systems literature, offline algorithmic evaluations frequently play a major role. There has also been work on evaluation methods themselves [61]. It is common to use offline analysis to pre-test algorithms in order to understand their behavior prior to user testing as well as to select a small set of candidates for user testing from a larger pool of potential designs. Since user trials can be expensive to conduct, it is useful to have methods for determining what algorithms are expected to perform the best before involving users. Offline evaluation is also beneficial for performing direct, objective comparisons of different algorithms in a reproducible fashion.

This section describes commonly used data sets for offline evaluations of recommender algorithms, the basic structure of these

evaluations, and metrics that are used to evaluate performance. It concludes with a brief discussion of online evaluations of recommender performance.

### 3.1 Data Sets

Many recommender systems are developed in particular contexts, and their evaluations will be on data sets relevant to that context or on internal data sets. There are, however, several data sets that are publicly available and are widely used in evaluating recommenders. They form a basis on which the raw numeric performance of new algorithms can be compared against known performance of existing systems in a consistent environment, and can serve as a preliminary testing domain in building a system for which no directly relevant data is available.

One of the early publicly available data sets was the EachMovie data set. The DEC Systems Research Center operated a movie recommender system called EachMovie; when the system was shut down in 1997, an anonymized version of the data set was made available.<sup>1</sup> This data set consisted of 2.8 M user ratings of movies.

The MovieLens movie recommender, operated by GroupLens research, was bootstrapped from the EachMovie data set [35]. It has since made three data sets available: one with 100 K timestamped user ratings of movies, another with 1 M ratings, and a third containing 10 M ratings and 100 K timestamped records of users applying tags to movies.<sup>2</sup> The ratings in the MovieLens data set are user-provided star ratings, from 0.5 to 5 stars; older data sets have a granularity of 1-star, while newer ones have 1/2 star granularity.

The Jester data set is a set of user ratings of jokes collected from the Jester joke recommender system [50]. It consists of two data sets: one has ratings of 100 jokes from 73,421 users between April 1999 and May 2003, and the other has ratings of 150 jokes from 63,974 users between Nov 2006 and May 2009. These ratings are continuous ratings in the range  $[-10, 10]$ .<sup>3</sup> As a result of the “gauge set” required by the

---

<sup>1</sup> The EachMovie data set was later withdrawn in 2004.

<sup>2</sup> The MovieLens data sets are available at <http://www.grouplens.org/node/73>.

<sup>3</sup> The Jester data set is available at <http://eigentaste.berkeley.edu/dataset/>.

Eigentaste algorithm, a portion of the Jester data set is dense: there is a small set of jokes that almost every user has rated.

The BookCrossing data set is a collection of book ratings with some demographic information about users, collected from the book sharing and discussion site [bookcrossing.com](http://bookcrossing.com) [156].<sup>4</sup> It contains over 1.1 M ratings from 279 K users for 271 K books. The ratings are a mix of explicit real-valued (1–10) and implicit unary ratings.

The Netflix data set, made available in 2006 as a part of the Netflix Prize [15], has been widely used as a large-scale data set for evaluating recommenders. It consists of over 100 M datestamped ratings of 17 K movies from 480 K users, with some perturbation applied to the ratings in an attempt to preserve privacy. The data set was withdrawn in late 2009 after publicity surrounding research to de-anonymize anonymized data sets [41, 105].

There are also other data sets in circulation from various sources. Yahoo! Research operates one such collection, providing a variety of data sets collected from Yahoo! services.

### 3.2 Offline Evaluation Structure

The basic structure for offline evaluation is based on the train-test setup common in machine learning. It starts with a data set, typically consisting of a collection of user ratings or histories and possibly containing additional information about users and/or items. The users in this data set are then split into two groups: the training set and the test set. A recommender model is built against the training set. The users in the test set are then considered in turn, and have their ratings or purchases split into two parts, the *query set* and the *target set*. The recommender is given the query set as a user history and asked to recommend items or to predict ratings for the items in the target set; it is then evaluated on how well its recommendations or predictions match with those held out in the query. This whole process is frequently repeated as in  $k$ -fold cross-validation by splitting the users into  $k$  equal

---

<sup>4</sup>The BookCrossing data set is available at <http://www.informatik.uni-freiburg.de/~cziegler/BX/>.

sets and using each set in turn as the test set with the union of all other sets as the training set. The results from each run can then be aggregated to assess the recommender’s overall performance, mitigating the effects of test set variation [53].

This form of offline analysis has formed the basis of many evaluations of collaborative filtering, starting with [22]. Herlocker et al. [61] used this method as the basis for a comparative analysis of various evaluation methods and metrics.

Further refinements to the offline evaluation model take advantage of the temporal aspects of timestamped data sets to provide more realistic offline simulations of user interaction with the service. A simple temporal refinement is to use time rather than random sampling to determine which ratings to hold out from a test user’s profile [53]; this captures any information latent in the order in which the user provided ratings. Further realism can be obtained by restricting the training phase as well, so that in predicting a rating or making a recommendation at time  $t$ , the recommendation algorithm is only allowed to consider those ratings which happened prior to  $t$  [25, 53, 85]. This comes at additional computational expense, as any applicable model must be continually updated or re-trained as the evaluation works its way through the data set, but allows greater insight into how the algorithm performs over time.

### 3.3 Prediction Accuracy

When testing recommender algorithms that are based on predicting user preference against a data set that expresses real-valued user ratings (such as the star ratings in MovieLens or the continuous ratings in Jester), measuring the accuracy of predicted values with respect to the held-out ratings is a natural starting point for evaluating recommender output.

A straightforward method of measuring the recommendation quality is to measure the *mean absolute error* (MAE) [22, 58, 59, 112, 137], sometimes also called absolute deviation. This method simply takes the mean of the absolute difference between each prediction and rating for all held-out ratings of users in the test set. If there are  $n$  held-out

ratings, the MAE is computed as follows:

$$\frac{1}{n} \sum_{u,i} |p_{u,i} - r_{u,i}| \quad (3.1)$$

MAE is in the same scale of the original ratings: on a 5-star scale represented as by integers in [1, 5], an MAE of 0.7 means that the algorithm, on average, was off by 0.7 stars. This is useful for understanding the results in a particular context, but makes it difficult to compare results across data sets as they have differing rating ranges (an error of 0.7 is more consequential when ratings are in [1, 5] than when they are in [-10, 10]). The *normalized mean absolute error* (NMAE) [50] is sometimes used to address this deficiency. NMAE normalizes errors by dividing by the range of possible ratings ( $r_{\text{high}}$  and  $r_{\text{low}}$  are the maximum and minimum ratings in the system, respectively), resulting in a metric in the range [0, 1] for all rating scales:

$$\frac{1}{n(r_{\text{high}} - r_{\text{low}})} \sum_{u,i} |p_{u,i} - r_{u,i}| \quad (3.2)$$

NMAE results are harder to interpret in terms of the original ratings scale but are comparable across data sets with different ratings scales. They are therefore useful in measuring the impact of aspects of the data set on the performance of a recommender.

*Root mean squared error* (RMSE) [61] is a related measure that has the effect of placing greater emphasis on large errors: on a 5-star scale, the algorithm is penalized more for being 2 stars off in a single prediction than for being off by 1/4 of a star 8 times. It is computed like MAE, but squares the error before summing it:

$$\sqrt{\frac{1}{n} \sum_{u,i} (p_{u,i} - r_{u,i})^2} \quad (3.3)$$

Like MAE, RMSE is in the same scale as the original ratings. Famously, it was the measure chosen for the Netflix Prize: the \$1 M prize was awarded for a 10% improvement in RMSE over Netflix's internal algorithm. It can also be normalized for the rating scale like NMAE by multiplying by  $1/(r_{\text{high}} - r_{\text{low}})$ .

Equations (3.1)–(3.3) all describe measuring the accuracy across the entire data set at once. They can also be adapted to compute the average error for each user and aggregate the average errors to compute a score across the entire test set. Overall and per-user aggregation can result in different relative performance measures for algorithms. Aggregating by user provides a measurement of how users will experience the system, while overall aggregation gets at how the system will perform in general. Examining the error distribution by user or by item can be useful for identifying particular strong or weak areas for the recommender, either in user taste or item type.

All three of these prediction accuracy measures effectively measure the same thing. Which one to use depends on how the results are to be compared and presented (MAE vs. NMAE) or whether the additional penalty for large errors is desired ( $[N]MAE$  vs.  $[N]RMSE$ ). They are only useful, however, for evaluating a recommender on the *predict* task. They also have the drawback of treating errors equivalently across items, while the error on low-importance items is likely to have less impact on user experience than the error on popular or important items.

To compute a single metric of predictive accuracy, we recommend RMSE due to its increased penalty for gross prediction errors. If metrics need to be compared between data sets, it can be normalized in the same manner as NMAE.

Predictive accuracy metrics focus on the predict task, but can also be indicative of a recommender’s performance for recommendation as well. Koren [79] showed that, for a selection of algorithms, the ones with better RMSE performance also were more consistent at recommending movies users liked over randomly selected movies.

### 3.4 Accuracy Over Time

MAE and related metrics provide a static view of the predictive performance of a recommender system outside of time. Often, though, the performance of the recommender over time is interesting: does the recommender generate better results as more users use it? For individual users, how does accuracy change as they add more ratings?

This has led to the proposal of temporal versions of MAE and RMSE. Time-averaged RMSE [85] requires that predictions be performed in temporal sequence using only ratings before the current time  $t$ , as discussed by Gunawardana and Shani [53], and computes the RMSE over all predictions generated up to time  $t$ . Equation (3.4) shows this computation, with  $n_t$  being the number of ratings computed up through time  $t$  and  $t_{u,i}$  being the time of rating  $r_{u,i}$ .

$$\text{TA-RMSE}_t = \sqrt{\frac{1}{n_t} \sum_{p_{u,i}: t_{u,i} \leq t} (p_{u,i} - r_{u,i})^2} \quad (3.4)$$

Lathia [84] also proposes windowed and sequential versions of this metric, restricting the time-averaging to ratings with in a window or simply computing the average error in a window, respectively. Burke [25] proposes Temporal MAE, an MAE version of Lathia's sequential RMSE, and another metric called Profile MAE which provides a user-centric view of error over time. Profile MAE computes how individual users experience the recommender, showing how the error in the predictions changes as they add ratings. It is also computed by considering each rating in temporal order, but averages error over the number of items in the user's profile at the time of prediction rather than by user or time window.

### 3.5 Ranking Accuracy

For contexts where ranked lists of items are produced and item rankings can be extracted from the user's rating data, the algorithm's capacity to produce rankings that match the user's can be measured.

Two ways of measuring this are with Pearson correlation or Spearman rank correlation coefficients. As noted by Herlocker et al. [61], however, the Spearman rank correlation metric suffers when the recommender makes distinctions between items that the user ranks at the same level. These metrics also penalize inaccuracy in the tail of the recommendation list, which the user will likely never see, to the same degree as inaccuracy in the top-predicted items.

Another metric for evaluating ranking accuracy is the *half-life utility* metric [22]. It measures the expected utility of a ranked

recommendation list, based on the assumption that users are more likely to look at items higher in the list; this assumption is reasonable for many real systems such as e-commerce sites. It requires two parameters: a half-life  $\alpha$ , such that the probability of a user viewing a recommendation with rank  $\alpha$  is 0.5, and a default rating  $d$ .  $d$  should be selected to indicate neutrality, providing neither benefit for the user; it can be the user's mean rating, the system's overall mean rating, or some fixed neutral/ambivalent point in the rating scale. In unary domains, where this metric is most frequently applied,  $d$  can be 0 for “not purchased”, with purchased items having a rating  $r_{u,i} = 1$ .

The half-life expected utility  $R_u$  of the recommendation list for a user  $u$  is defined in Equation (3.5).  $k_i$  is the 1-based rank at which item  $i$  appears.

$$R_u = \sum_i \frac{\max(r_{u,i} - d, 0)}{2^{(k_i-1)/(\alpha-1)}} \quad (3.5)$$

In order to measure the performance of a recommender across users, this metric is normalized by the maximum achievable utility  $R_u^{\max}$  for each user, resulting in a value in  $[0, 1]$  representing the fraction of potential utility achieved.  $R_u^{\max}$  is the utility achieved by listing the best items for the user in order. The normalization is needed to compensate for different users having different potential utilities; one way this can happen is as a result of some users having more purchases to predict than others. The overall score  $R$  is given by Equation (3.6).

$$R = \frac{\sum_u R_u}{\sum_u R_u^{\max}} \quad (3.6)$$

Half-life utility can be difficult to apply and compare, as it depends on choosing  $\alpha$  and  $d$  appropriately. The work of Agichtein et al. [2] is helpful for approximating  $\alpha$  in the absence of domain-specific data; their analysis of search query logs shows that 50% of search result clicks are on the top 5 items, making  $\alpha = 5$  a reasonable starting point.

Half-life utility suffers from the further drawback of penalizing recommendations of mildly-disliked and adamantly hated items equivalently if  $d$  is a rating value indicating ambivalence about an item [61]. However, in domains suggesting natural values for the parameters, its

focus on providing the highest-utility items at the beginning of the list make it a powerful tool for measuring the ability of a recommender to provide maximum value to users. Variants of it can be particularly useful if more nuanced utility functions are available to substitute for  $\max(r_{u,i} - d, 0)$ .

### 3.6 Decision Support Metrics

Recommender performance, particularly in unary domains such as purchase histories, can also be evaluated in the precision-recall framework of information retrieval derived from statistical decision theory [126]. This framework examines the capacity for a retrieval system to accurately identify resources relevant to a query, measuring separately its capacity to find all relevant items and avoid finding irrelevant items. Figure 3.1 shows the *confusion matrix*, depicting the relationship of these concepts. Recommenders are evaluated with retrieval metrics by operating in a unary purchase domain, holding some items out of the user’s profile, and considering the held-out purchased items to be relevant to a query consisting of their remaining profile [22, 61, 131]. These metrics can also be applied in non-unary ratings domains by using a rating threshold to distinguish between liked and disliked items.

To be applied robustly, all of these measures require a fully-coded corpus where every item is known to be either relevant or irrelevant to the query under consideration. In recommender evaluation, however, that data is not generally available — if an item has not been purchased, it could be because the user does not like the item, or it could be because they were unaware of it. If the recommender returns an item outside the held-out target set, it still could be a relevant item. Care must be taken, therefore, when designing and evaluating precision-recall assessments

|                      | <b>Relevant</b> | <b>Irrelevant</b> |
|----------------------|-----------------|-------------------|
| <b>Retrieved</b>     | TP              | FP                |
| <b>Not retrieved</b> | FN              | TN                |

Fig. 3.1 Retrieval confusion matrix.

|               | Relevant | Irrelevant |               | Relevant | Irrelevant |
|---------------|----------|------------|---------------|----------|------------|
| Retrieved     | TP       | FP         | Retrieved     | TP       | FP         |
| Not retrieved | FN       | TN         | Not retrieved | FN       | TN         |

(a) Precision:  $\frac{TP}{TP+FP}$

(b) Recall:  $\frac{TP}{TP+FN}$

Fig. 3.2 Precision (a) and recall (b) in the confusion matrix.

of recommender systems; nevertheless, it can be a useful framework for understanding recommender performance.

The core metrics in precision-recall evaluations are the *precision P*, the fraction of items returned by the recommender that are purchased ( $\frac{TP}{TP+FP}$ ), and the *recall R*, the fraction of purchased items returned by the recommender ( $\frac{TP}{TP+FN}$ ) [126]. Figure 3.2 shows how these metrics relate to the confusion matrix. Since recommenders and retrieval systems typically return ranked lists, these are usually computed for some fixed recommendation list length  $N$  or sometimes for a threshold score. For a given system, precision and recall are inversely related and dependent on  $N$ , so comparing results between recommenders can be cumbersome [61]; however, the combination can provide more insight into the particular behavior of an algorithm than single-number measures. Different user tasks require different tradeoffs between precision and recall — a user looking for a movie recommendation likely only cares that they get a good movie (high precision), while a lawyer looking for legal precedent needs to find all relevant cases (high recall). Herlocker et al. [61] classify these two disparate needs as “Find Good Items” and “Find All Good Items”.

Precision and recall can be simplified into a single metric, the  $F_1$  metric [124, 153], which has been used to evaluate recommender systems [131]:

$$F_1 = \frac{2PR}{P + R}$$

$F_1$  blends precision and recall with equal weight. The resulting number makes comparison between algorithms and across data sets easy, but does not facilitate more nuanced comparisons where one of the two measures is more important than the other.

|               | Relevant | Irrelevant |
|---------------|----------|------------|
| Retrieved     | TP       | FP         |
| Not retrieved | FN       | TN         |

(a) Sensitivity:  $\frac{\text{TP}}{\text{TP} + \text{FN}}$

|               | Relevant | Irrelevant |
|---------------|----------|------------|
| Retrieved     | TP       | FP         |
| Not retrieved | FN       | TN         |

(b) Specificity:  $\frac{TN}{TP+FN}$

Fig. 3.3 Sensitivity (a) and specificity (b) in the confusion matrix.

Precision and recall also ignore the lower right quadrant of the confusion matrix — the true negatives, those items correctly identified as irrelevant or disliked. In order to incorporate that quadrant into a metric, a similar pair of metrics called *sensitivity* and *specificity* can be used [83]. Figure 3.3 shows these metrics. Sensitivity is equivalent to recall (the true positive rate) while specificity measures the fraction of irrelevant items correctly discarded (the true negative rate).

A related but more comprehensive comparison method is the *relative operating characteristic* (ROC, also called receiver operating characteristic) curve [142]. This curve is derived from sensitivity and specificity and plots sensitivity against the complement of the specificity. As the length of the recommendation list increases, more items are included, until the entire item set (including all relevant and irrelevant items) has been returned. ROC curves effectively describe the relationship of precision and recall across the range of their tradeoff, showing how the recall of the system increases as precision requirements decrease. Figure 3.4 shows some sample ROC curves. A recommender which randomly returns items will, on average, plot a straight diagonal line; curves above the line are better than random. The further to the upper-left corner of the plot the curve reaches, the better the recommender performs. A perfect recommender (which returns all target items before any irrelevant items) will plot a vertical line to (0,0,1.0) and a horizontal line to (1,0,1.0).

The ROC curve can also be used to derive a single numeric measure of performance by computing the area under the curve, called Swets' A-measure or AUC; a perfect recommender will achieve an AUC of 1. AUC is limited, as a given recommender system will operate a particular ratio of precision to recall or will move along the curve as the user explores longer recommendation lists. It is useful, however, for

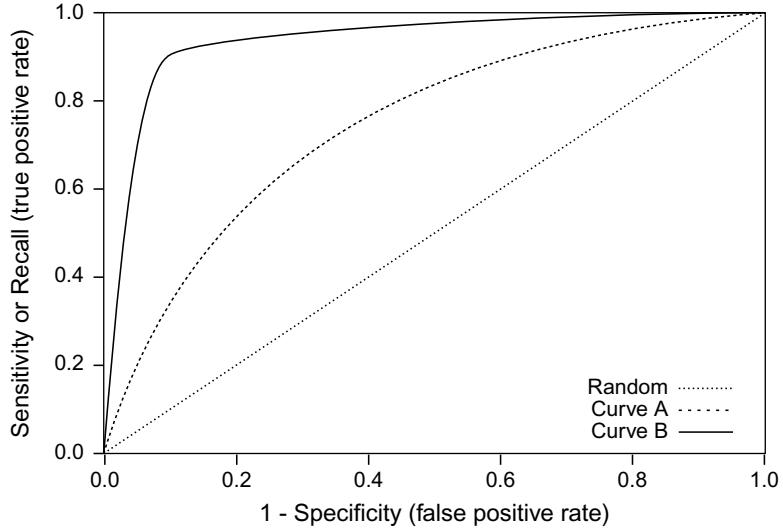


Fig. 3.4 Sample ROC curves.

quantifying the tradeoff in precision necessary to increase recall with a particular system, measuring in some sense the “efficiency” of the recommender.

ROC curves are good for performing detailed assessments of the performance of particular algorithms. Comparing them, however, is challenging, particularly for large numbers of algorithms. Therefore, precision/recall,  $F_1$ , and AUC are useful for assessing relative performance of various algorithms in decision support contexts at the expense of oversimplifying recommendation performance.

### 3.7 Online Evaluation

Offline evaluation, while useful for comparing recommender performance and gaining understanding into how various algorithms behave, is limited in a variety of ways [88, 89, 98]. It is limited to operating on data regarding past behavior — a leave-N-out recommender trial measures only the ability to recommend items the user found somehow. Further, there is a selection bias in historical rating data, as users are more likely to consume and therefore be able to rate items

they perceive to be good. Time-based methods compensate for this somewhat, but offline evaluations are incapable of measuring the recommender’s ability to recommend good items that the user did not eventually find somehow. They also cannot measure how users respond to recommendations or how recommendations impact business concerns. Recommendation algorithms with similar numeric performance have been known to return observably different results [97, 146], and a decrease in error may or may not make the system better at meeting the user’s needs. The only way to accurately assess these factors is to do an online evaluation — to try a system with real users.

Online evaluations come in a variety of forms. Field trials — where a recommender is deployed in a live system and the users’ subsequent interactions with the system are recorded — enable the designer to see how users interact with the recommender and change their behavior in “real” environments. Since many recommenders are provided in web-based contexts, page views, interface clicks, rating behavior, and purchase behavior can be measured and used to assess the impact of the recommender. Field trials, frequently in the form of A/B tests, are commonly used in commercial recommender deployments where sites have already established user bases and can test a new algorithm or other site change on a small subset of their users [76, 77]. Amazon and Google are well-known for using A/B testing in refining and evaluating their services. Field trials can also be used to measure other aspects of system performance such as recommender accuracy in response to a change in the system [117]. When combined with user surveys, they can be used to assess user satisfaction and other subjective qualities. Recommender effectiveness with respect to business concerns can also be measured in field trials using a variety of measurements used to evaluate marketing campaigns [31].

Virtual lab studies are another common method for evaluating recommender systems. They can be used without access to a live user base, and are therefore useful for testing new ideas for which no deployed application exists or is available. They generally have a small number of users who are invited to participate, and can take the form of interaction with a full application [19] or a survey structured around the recommender [29, 39, 118, 146]. Virtual lab studies, while they provide

a great deal of flexibility and control in designing an experiment for testing a recommender, create an artificial interaction that may affect the applicability of the results to real applications.

Traditional in-lab studies are also used to evaluate recommender systems [141], but virtual lab studies and field trials are generally more common.

Many uses of online evaluation, such as measuring accepted recommendations, page views, conversions, or sales lift, are well-understood. Well-constructed surveys can help get at many other aspects of recommender quality and utility. Determining ways to measure more subtle aspects of recommender quality and user satisfaction is the subject of ongoing research. Shani and Gunawardana [135] describe many aspects of online and offline evaluations, and more work is needed to come up with good ways to assess recommender's ability to meet user's needs. *User Information Needs* discusses in more detail what some of these needs are and surveys research on addressing them.

# 4

---

## Building the Data Set

---

In order to provide recommendations to users, simply having a recommender algorithm is not sufficient. The algorithm needs a data set on which to operate — a ratings matrix (or its functional equivalent) must somehow be obtained from the users of a system. Brusilovsky [23] provides a summary of how this is necessary in the broader space of adaptive user experiences: the system must collect data about users in order to model their preferences and needs and ultimately provide them with a customized experience (in the present case, customized recommendations or preference predictions).

The need for preference data can be decomposed into two types of information needs. The system needs to know something about users' preferences: what items (or types of items) does each user prefer? It also needs to know something about items: what kinds of users like or dislike each item? This breakdown is perhaps clearest when thinking of matrix decomposition models for recommendation: the user-item preferences are factored into a set of characteristics, user preferences for those characteristics, and those characteristics' applicability to various items. These needs are also present in other models to varying extents: the item-item CF model, for instance, needs to know what items are

liked by the same users (a combination of “what users like this item?” and “what other items are liked by these users?”) as well as the current user’s preferences.

These information needs come up in the recommender systems literature primarily under the guise of the *cold-start problem*. The cold-start problem, in general, is the problem of providing recommendations when there is not yet data available on which to base those predictions. It takes two flavors:

- *Item cold-start*, where a new item has been added to the database (e.g., when a new movie or book is released) but has not yet received enough ratings to be recommendable [133].
- *User cold-start*, where a new user has joined the system but their preferences are not yet known [34].

The bootstrap problem, where a system has no information about any user preferences, is an extreme intersection of both aspects of the cold-start problem.

Not only does a recommender system need data, it needs that data to be of high quality in order to provide useful recommendations. If it does not accurately know user preferences, its ability to generate useful recommendations can suffer — prediction error is lower-bounded by the error in user ratings [62]. Rating error ( $|r_{u,i} - \pi_{u,i}|$ , the difference between users’ ratings and their true preferences) can be divided into two categories: natural noise, where user ratings differ from true preference due to error in the rating collection process, and malicious noise, discrepant ratings introduced for the purpose of manipulating the system’s recommendations or predictions [109]. In this section, we are only concerned with natural noise; malicious noise will be taken up in *Robustness* when we discuss security implications of recommender systems.

## 4.1 Sources of Preference Data

Preference data (ratings) comes from two primary sources. Explicit ratings are preferences the user has explicitly stated for particular items. Implicit “ratings” are inferred by the system from observable user activity, such as purchases or clicks.

Many recommender systems, in both academic and commercial settings, obtain ratings by having users explicitly state their preferences for items. These stated preferences are then used to estimate the user's preference for items they have not rated. From a system-building perspective, this is a straightforward method and avoids potentially difficult inference problems for user preferences. It suffers, however, from the drawback that there can, for many reasons, be a discrepancy between what the users say and what they do.

Preferences can also be inferred from user behavior [106]. In the Usenet domain, this has been examined by using newsgroup subscriptions [69] and user actions such as time spent reading, saving or replying, and copying text into new articles [104]. Konstan et al. found a substantial correlation between the time a user spent reading a Usenet article and his/her rating of that article [78], further supporting the idea that observable user behavior can be a useful source of preference information. Consumption behavior has also been extended beyond reading domains: while not a collaborative recommender system, the Intelligent Music Management System plugs in to a user's music player and automatically infers the user's preference for various songs in their library as they skip them or allow them to play to completion [52]. The PHOAKS system mined Usenet articles for another form of implicit preference data: mentions of URLs. It used the frequency with which URLs are mentioned in Usenet articles to provide recommendations for web resources [63]. In e-commerce domains, implicit feedback in the form of page views and item purchases is the primary source of preference data for recommendation [87].

System designers and analysts must be careful when using implicit data to understand the mapping between system-visible user identities (frequently accounts) and actual people. Users of e-commerce systems, for example, will often purchase items as gifts for others. Those purchases and related activity do not necessarily communicate anything about the user's tastes. Users will also share accounts on many systems; a family may have a single Amazon.com account, for instance, thus providing the system with an ill-defined aggregate of several people's preferences. Netflix deals with this problem by allowing each account to have multiple profiles with separate ratings, queues, and recommendations.

O'Mahony et al. [109] argue that observed or inferred preference information is expected to be noisier than explicitly provided data. It is possible, however, that inference can build a more accurate long-term profile of a user's taste than they themselves can articulate.

Explicit and implicit rating data are not mutually exclusive. Systems can incorporate both into the recommendation process. Most recommender systems are capable of collecting some implicit information at least through page views. As rating mechanisms become increasingly prevalent in a variety of web sites, particularly e-commerce sites, further hybrid data becomes available to the operator of those services. Amazon.com makes use of both explicit and implicit data in providing its various recommendations.

Rating data can also be obtained from external sources, although this data may not be identifiably connected to users of the recipient system. In systems with few users or brand-new systems facing the bootstrapping problem, external data sources can be particularly valuable in getting initial preference data to actually be able to make recommendations. MovieLens did this by bootstrapping with the EachMovie data set [35].

Preference data can also be taxonomized based on its relationship to the user's experience of the item. Table 4.1 shows how the experience relation relates to implicit and explicit data with some examples.

- *Consumption ratings* are provided when the user has just consumed (or is consuming) the item in question. These ratings are particularly common in online streaming media environments, such as the Pandora music recommender or NetFlix instant streaming.

Table 4.1. Types of preference data.

|          | Consumption  | Memory   | Expectation                            |
|----------|--|--|--|
| Explicit | Pandora song ratings,<br>GroupLens Usenet<br>article ratings | Netflix or MovieLens<br>movie ratings                        | House advertisement<br>ratings         |
| Implicit | Streaming video<br>watch time                                | Views of discussion<br>pages related to<br>previous purchase | Dwell time on new car<br>advertisement |

- *Memory ratings* are provided based on the user’s memory of experiencing the item. When a user watches a movie at the theater and then rates it on MovieLens or MoviePilot, they are providing a memory rating.
- *Expectation ratings* are provided when the user has not yet experienced the item (and may never experience it) but is willing to provide a rating anyway. One example is rating housing options — the user will likely only have the means and opportunity to truly experience one housing situation, but may provide ratings on other house advertisements indicating their level of appeal to the user.

In some ways, consumption ratings are the most reliable, as the item is fresh in the user’s mind. Memory ratings are also based on experience, so the user has a fuller set of data on which to base their rating than in the expectation case, but their impression of the item may not be accurately remembered or may be influenced by various external factors [68]. Expectation ratings are the most limited, as the user has not experienced the item and thus is rating it based only on the information presented about it combined with whatever prior knowledge they may have. They can still be useful, however, particularly in high-cost domains.

## 4.2 Rating Scales

Mapping user preference to a rating scale is not an easy task. While utility theory shows that there is a real-valued mapping for the preferences of any rational agent [149], it is not necessarily easy for users to express their preferences in this way. Further, preference is not a function only of the item: contextual factors such as time, mood, and social environment can influence a user’s preference, so provided ratings may contain information about the context in which the user experienced the item and, in the case of memory ratings, intervening events or changes of mind as well as their preference for the item itself.

Recommender systems over the years have used (and continue to use) a wide variety of different scales and methods for collect-



Fig. 4.1 The MovieLens 5-star rating interface. The page also contains a legend, defining 1 to 5 stars as “Awful”, “Fairly Bad”, “It’s OK”, “Will Enjoy”, and “Must See”.

ing ratings [34]. Many of these systems are a numeric scale of some kind: GroupLens used a 5-star scale [119], Jester uses a semi-continuous  $-10$  to  $+10$  graphical scale [50], and Ringo used a 7-star scale [137]; Figure 4.1 shows a 5-star scale as deployed in MovieLens. These scales are appealing to the system designer, as they provide an integral or real-valued rating that is easy to use in computations. They also benefit from the body of research on Likert scales in the survey design and analysis literature [9, 43, 45].

Explicitly numeric scales are not the only scales in use. Some systems use a “like” / “dislike” method — the Pandora music recommender provides thumbs-up and thumbs-down buttons which can be used to request more songs like the current song or ban the current song from the station. The Yahoo Launch music player used a real-valued “like” rating (0–100) in combination with a “ban” button [34]. MovieCritic.com had users rank films and derived ratings from these rankings.

Cosley et al. [34] found that users prefer finer-grained scales when rating movies, and that scale had little impact on user ratings (users rated items consistently even when using different scales). That work also found that the rating scale had little impact on prediction accuracy. Therefore, using fine-grained scales (such as allowing 1/2 star ratings) makes sense as users prefer it and it does not harm recommender accuracy. There is little research into the particular impact of rating interfaces in other contexts. Intuitively, simple like/dislike scales seem useful in systems such as real-time music recommenders where

the user actually experiences the item rather than receiving a list of suggestions. It is important to include both like and dislike, however; in a study of user rating behavior for movie tags, Sen et al. [134] found that users with both “like” and “dislike” buttons provided more ratings than users with only one rating option, and users who had a “dislike” button but no “like” button provided far more ratings than users whose only option was “like”.

When implicit ratings are collected, the observed user events must be translated into computationally useful estimators of preference [106]. Purchases, page views, and data such as the mention data used by PHOAKS are frequently treated as unary ratings data: purchased items have a “rating” of 1 and other items have unknown ratings [71]. With unary ratings, all that is known is that the user purchased a particular set of products: the fact that the seller does not have any information on a user’s interest in other products does not imply that the user does not like those products. They may be unaware of the existence of unpurchased products (a situation the recommender likely aims to change), or they may have purchased them from a different vendor. Therefore, while a purchase can be considered a vote for a product, a lack-of-purchase alone may not be considered a vote against.

Other forms of data, such as the time spent on a page, also provide useful data points for inferring user preference. In the Usenet domain, time spent reading an article correlates strongly with the user finding that article interesting [78, 104]. It can be beneficial to consider a variety of observable user activities, particularly to discern the user’s interest in an item they have not purchased in a commerce domain.

### 4.3 Soliciting Ratings

Beyond the input method used for providing ratings, there are also questions to consider in selecting which items to ask or encourage users to rate. To optimize individual user experience, a recommender system should provide quality recommendations with as enjoyable a rating experience as possible; in some cases, that may mean minimizing the effort required from the user, but requiring work users perceive as meaningful can result in higher user loyalty [96]. It should also provide an

enjoyable experience for the user. Therefore, when a new user joins the system, conducting the initial “interview” strategically, aiming to maximize the information gained about the user’s tastes for the items presented for rating, can provide a smooth initial user experience. This is not the only consideration, however — recommender systems have multiple users, and collaborative filtering depends on knowing as many users’ preferences for items as possible. Therefore, interview strategies can also attempt to improve the system’s ability to serve the user base as a whole by asking users to rate items for which few ratings have been collected. Further, new users are not the only entities about which information needs to be collected. New items also present a problem, and the system has the opportunity to strategically present them to existing users to gain an understanding of their appeal profile.

Information and decision theory provide some insight into how this problem can be approached. Value-of-information (VOI) calculations can be used, based on estimated improvement in prediction accuracy, to ask the user to rate items that provide the system with the highest expected improvement in quality [112]. Entropy-based measures can also be used, determining which items will likely provide the system with the most information about the user’s taste [117]. The user’s preference for a movie with high variance in ratings tells the system more about that user’s taste than their preference for a movie with more uniform ratings. Entropy alone, however, is not sufficient, as there may be many items with high information value that the user is unable to rate. Therefore, a probability that the user will be able to rate the item must also be factored in to the decision process; Rashid et al. [117] used the popularity of items, asking the user to rate items that were both popular and had high information value. The structure of the resulting formula is similar to value-of-information, maximizing the expected information gain.

The cost of consuming items is also a factor in the ability of a system to solicit user ratings and fill in user taste profiles. Movies, for example, have a higher cost of consumption than songs — even if the media is free, it takes around 2 hours to watch a movie while many songs last only 3–4 minutes. Therefore, a movie recommender is likely to depend on finding movies users have seen in order to collect ratings

(thus relying predominantly on memory ratings). A music recommender can have more flexibility — systems like Pandora or Launch.com can actually play the song for the user to get their opinion of it rather than just giving them a title and artist. Similarly, Jester has users rate jokes based on reading the joke; the small cost of reading a joke allows Jester to present the same starting set of jokes to each user to build an initial version of their preference profile [50]. These systems are therefore able to make greater use of consumption ratings. Some domains, such as house or car purchases, have potentially very high cost, where items are expensive and their true utility is not fully known until they have been used for an extended period of time. Systems in such domains are therefore either unable to use ratings or depend primarily on expectation ratings.

A further consideration in soliciting ratings is whether or not to show the user's predicted preference in the rating interface. Cosley et al. [34] found that predicted preferences influence users' ratings, and that some expert users are aware of this influence. System designers should be aware of this and careful to consider what impact it will have on their users.

There has also been work looking more directly at user motivation and willingness to provide ratings. Harper et al. [56] provide an economic model for user behavior in providing ratings. This incorporates some issues already discussed, such as the time and effort needed to provide a rating, and provides a framework for reasoning about how users are motivated to interact with recommender systems. Avery and Zeckhauser [11] argue that external incentives are needed to provide an optimal set of recommendations and that market-based systems or social norms can provide a framework for promoting user contribution to the rating data.

#### 4.4 Dealing with Noise

The process of expressing preference as ratings is not without error — the ratings provided by users will contain noise. This noise can result from normal human error, confounding factors in the rating process (such as the order in which items are presented for rating), and other

factors in user experience (e.g., a noisy theater detracting from the enjoyment of an otherwise-good movie). Detecting and compensating for noise in the user’s input ratings therefore has potential to result in better recommender systems.

Natural noise in ratings can be detected by asking users to re-rate items [5, 34, 62]. These studies all found that users’ ratings are fairly stable over time. Amatriain et al. [5] further found that moderate preferences — 2 or 3 on a 1–5 scale — are less stable than extreme like or dislike, and performed a more thorough analysis of the re-rating data to determine that ratings are a reliable method of measuring user preferences by the standards of classical test theory. Once noisy or unstable preferences have been detected, they can be discarded to improve the recommender’s prediction accuracy [6].

O’Mahony et al. [109] proposed detecting and ignoring noisy ratings by comparing each rating to the user’s predicted preference for that item and discarding ratings whose differences exceed some threshold from the prediction and recommendation process. This approach, however, makes the assumption that all high-deviance ratings are a result of user error rather than aspects of user preference for which the system cannot yet account, and discarding such ratings may inhibit (at least temporarily) the system’s ability to learn that aspect of the user’s taste.

Some recommendation algorithms have noise built into their user ratings model. Personality diagnosis explicitly assumes that user-provided ratings have Gaussian noise, being drawn from a normal distribution whose mean is the user’s true preference [112]. Analyzing ratings with this model in a naïve Bayesian framework allows data from other users and the smoothing effect of probabilistic inference to compensate somewhat for noise in individual ratings.

# 5

---

## User Information Needs

---

Recommender systems do not exist in a vacuum. Treating recommendation abstractly as mathematical problem, aiming primarily to improve offline evaluations with prediction accuracy metrics such as RMSE, ignores the broader context of use and is not necessarily measuring the impact these systems have on their users. Re-grounding recommender systems in user needs can have a profound impact on how we approach the field. Anchoring both the design of the recommender system itself and the evaluation strategy used to measure its effectiveness to a detailed understanding of user goals, tasks, and context can enable us to build systems that better serve their users [74, 99].

Users use a recommender system for some purpose. With systems like GroupLens and PHOAKS, that purpose can be to more easily filter through high volumes of articles and find interesting new resources. Movie recommenders can help users find new movies and choose films to watch. A joke recommender can provide entertainment. In each case, we can consider the recommendations to have some utility to the user. User needs can also extend beyond the recommendation list — some users interact with recommender systems for the purpose of self-expression, with the rating process rather than the resulting recommendations being their desired end [56, 61]. Evaluation metrics are useful, therefore, to the extent that

they map to (or at least correlate with) the user’s utility derived from the system’s output (recommendations and predictions) and the overall experience it provides. Task- and need-driven user-based studies are needed to determine what factors actually do affect the system’s ability to meet user needs and improve the user experience.

## 5.1 User Tasks

The classical recommender tasks of *predict* and *recommend* can be re-cast in terms of user needs as “estimate how much I will like an item” and “find items I will like”, respectively [61]. User needs can also take more nuanced manifestations. For example, users can use a recommender system to find new items they may like (*introduction*) or to recall previous items they enjoyed (*reuse*); systems like the Pandora music recommender are built to meet these two needs in balance (users want to discover new music while also listening to music they know they like). Users can also be interested in merely exploring the item space (*explore*, the “Just Browsing” task of Herlocker et al. [61]) rather than supporting a particular decision (*make decision*). Some users have the goal of determining the recommender’s credibility (*evaluate recommender*) — they may wish to see how the recommender does at estimating their preference for items they know well in order to determine how much they trust its unfamiliar recommendations (or how much time they want to invest in training the system). By supporting this last task, a balance of familiar and unfamiliar items can be important in a developing long-term relationships between users and recommenders [99].

The type of interaction the user has with the recommender also impacts how it should perform. Does the user treat the recommender as an information source to be searched, or as a decision support tool? Even when using a recommender for decision support, users still have differing goals. In some cases they may be primarily interested in exploring the space of options. They may turn to the recommender to provide them with a candidate set from which they will choose. In other cases, they may want the recommender to actually make the selection. A movie recommender or an e-commerce site is likely to be used to explore or determine a candidate set; the user generally makes the

final decision on which movie to see. Real-time music recommenders such as Pandora, however, select a song for the user to hear next, providing the user with a means for critiquing the decision but not often do not allow the user directly select from a set of items.

Even recommenders with equivalent numerical performance can have qualitative differences in their result lists [146]. McNee et al. [99] call these *personalities*, and choosing an algorithm whose personality matches the user’s needs can provide greater user satisfaction.

## 5.2 Needs for Individual Items

Thinking about prediction from the perspective of user perception has led to the adoption of some common evaluation metrics: RMSE’s increased penalty for high error and other variants on MAE designed to distinguish high error both stem from the assumption that users are likely to forgive a small error in preference estimation (such as mispredicting by half a star), while gross errors (predicting a 3-star movie as a 5-star) will more significantly hamper user experience. It is also frequently more important to be accurate at the high end of the scale: if the system correctly predicts that the user will dislike an item, does it matter how much they will dislike it?

Another need users may have in their use of the recommender system is *serendipity* [98, 116]. Serendipity is the ability to recommend items that the user likes but does not expect; a movie, perhaps, that they never would have thought of liking but, once recommended, turned out to be enjoyable. Since users frequently use recommenders to learn about and try things that they would not otherwise know about, serendipity is generally a desirable characteristic of a recommender.

Related to serendipity is the factor of *risk* — what are the odds the user will like the item? If the system is recommending items that the user does not expect, it could be wrong sometimes and recommend a bad item. In some contexts, users may be better served by a recommender which is more aggressive in exploring the boundaries of the user’s preference space in hopes of achieving serendipity, while other contexts may merit a more conservative system that prefers to avoid the chance of a bad recommendation.

Risk can be modelled by the system’s confidence in its prediction, but that still leaves open the question of how much risk to take. Is the user more interested in finding potentially great items, or will the user accept merely good items in order to avoid bad ones, even if that involves missing potentially valuable recommendations? This is a function of both the user’s desires and the characteristics of the domain. In low-cost domains, particularly interactive ones such as music recommenders, the cost of following a bad recommendation is low, so a recommender with riskier behavior may be more appropriate than in a higher-cost domain.

The issue of new versus old items arises here as well. If the user’s task is “find a movie to go see this weekend”, they are likely not interested in a movie they have seen before. But if it is “play some music for my commute”, a blend of old and new may well be appropriate. Careful analysis of who the users are and what they expect from a system will provide insight into how to design the recommendation strategy.

### 5.3 Needs for Sets of Items

Many recommenders do not just recommend a single item to the user. They present a list, usually ranked, of items for the user to consider. Even recommenders which, at first glance, deal with individual items at a time may have a sense of the set or the whole: a music recommender plays individual songs in turn for a user, but the user perceives a sequence of songs as well as the individual selections [40].

At a basic level, the rank-accuracy metrics fit in line with viewing recommendation from a user perspective, as the exact preference prediction frequently does not matter when the user is only presented with a ranked list of choices. Half-life utility [22] adds the additional factor of modeling whether the user will even see the item in question, making it useful for assessing utility to the user in search-like contexts.

When users receive a list of recommended items, they do not experience it as a list of individual items but rather experience the list as an entity itself. It can well be that the best 5 items do not necessarily make the best 5-item recommendation list.

Research on research paper recommendations has shown that different algorithms produce lists that are better-suited for different goals: some algorithms produce lists better-suited for finding novel or authoritative work on a topic, while others produce lists that are better introductory or survey reading lists [146]. Users could discern differences between lists produced by different algorithms, including varying applicability to different tasks.

Perhaps the most prominent potential need users have when it comes to sets of items is that of diversity. Ali and van Stam [4] noted that, with standard collaborative filtering technology, a TV viewer who enjoys both science fiction shows and boating programs, but who has seen and rated more science fiction, may never receive boating recommendations. This domination of a profile by a single genre can reduce the recommender's capacity to satisfy the user needs across their multifaceted interests.

Diversity in recommendations can be desirable from two standpoints. First, users may have a variety of tastes, some expressed more strongly than others, and benefit from the recommender being able to recommend items across the breadth of their preferences. In many cases, the recommender should not only span their interests but also be able to push at the edges of the user's known interests with the hopes of introducing them to material that they enjoy but would not have found otherwise. Immediately, it needs to avoid pigeonholing users — the science-fiction and boating TV viewer of Ali and van Stam's portfolio effect may want to watch a good boating show in a Saturday afternoon, and the recommender may have an insufficient sense of diversity to provide one.

Second, excessively narrow recommendations have the potential to fragment communities into small pockets with little in common, with the rate-recommend cycle reinforcing an increasingly focused taste at the expense of breadth. Societally, this balkanization can decrease communication and interactions between groups of people [147].

Diversity is not always a desirable aim, however. Some user needs are better met by a more focused set of recommendations, and the need for diversity can change over time. Consider buying a house: in

the early stages, recommending a wide variety of styles, prices, and neighborhoods could be a good strategy. As the user looks at houses and sees what they do and do not like, narrowing the field of recommendations to a more focused set of options can help them to make their final decision. In the latest stages, it can be desirable to recommend several effectively substitutable options in case one is unavailable. Again, understanding the user's needs and relationship to the system is key to determining the role diversity should play in recommendation.

To add diversity to a recommendation list, Ziegler et al. [156] proposed “topic diversification”, a method which discards some items of a recommendation list in favor of others to increase the diversity of items within a recommendation list; users found this to more comprehensively capture their reading interests in a study using the BookCrossing data set.

Another recent and promising approach for optimizing the variety of choices presented to the user in a recommendation set is regret-based recommendation [148], a recommendation algorithm based on utility theory. This method uses a model of the user's utility to recommend a set of items with minimal expected loss. The basic idea behind its operation is that a recommendation set that contains only comedies will have greater loss if the user is in the mood for an action movie than a set containing a mix of comedies and action movies. This framework is not just applicable to diversity, but can be adapted to address a host of whole-list concerns.

Key to both of these algorithms is that they consider the set of recommended items as a whole rather than simply picking the best individual recommendations. Seeking to maximize the expected utility of a set of recommended items rather than focusing on maximizing the user's interest on a per-item basis is a step forward in improving user experience across diverse tastes.

In order to understand and reason about diversity in recommendations, it is useful to understand more generally what diversity is and how to measure it. Harrison and Klein [57] identify three types of diversity studied in operations research: *separation*, where the members of a set are at different positions on a linear continuum; *variety*, where members of a set are members of different classes; and *disparity*, where

some members of a set have a higher status or position or are otherwise superior to other members. Separation and variety seem to be particularly relevant to assessing recommendation and preference: movies can be more or less violent (separation), and they can be of differing genres or from different directors (variety). Harrison and Klein's review also provides metrics for measuring the various types of diversity: standard deviation and Euclidian distance can be used to measure separation, and Blau's index and Teachman's entropy index are both useful for measuring variety among classes. Ziegler et al. [156] used the average pairwise similarity of items in a recommendation set, with proximity in a hierarchical taxonomy of the item space as the similarity metric, to measure the similarity (lack of diversity) within recommendation lists.

Measuring diversity is useful in understanding the behavior of a particular system, but the extent to which the diversity of a recommendation list impacts that list's ability to meet a user's need is harder to measure. It is also not always obvious what forms of diversity relate to users' needs. Ziegler et al. [156] found that increasing the diversity across Amazon.com book categories improved user's assessment of the value of book recommendation lists they received up to a point, after which further diversification decreased perceived value. The nature and impact of diversity will vary across information domains and user tasks.

Another major concern in set recommendation is that of interaction effects. Some items, while good individually, are either particularly bad or particularly good together [55]. Accounting for these is a difficult problem, and package recommendation under constraints frequently becomes NP-hard [152]. However, recommendations often do not need to be perfect — approximation algorithms are often sufficient — and some relationships or properties that are difficult or impossible to compute can, in some cases, be determined by humans and fed back into the system [150].

## 5.4 Systemic Needs

Not only do users have particular needs for individual items and recommendation lists, they have needs from the system as well.

### 5.4.1 Coverage

While many user needs are satisfied by finding some good items, and missing some items does not cause a problem, there are other cases where the penalty for missing items is high — high recall is critical [61].<sup>1</sup> This can be the case in locating research literature, where it is undesirable to overlook prior work, but is even more critical in legal settings: missing prior work in a patent application or a relevant precedent while handling litigation can have drastic consequences.

The concept of *coverage* provides a useful framework for analyzing how a recommender algorithm does or does not exhibit this problem. Simply put, the coverage with respect to a user is the fraction of the total item domain which the algorithm could possibly recommend to the user (in typical collaborative filtering algorithms, this is the set of items for which predictions can be generated for the user). In a user–user recommender, the ability to recommend an item  $i$  is dependent on the user having potential neighbors which have rated  $i$ ; in an item–item recommender, it is dependent on the user having ratings for items in  $i$ 's list of neighbors. Other algorithms can have similar limitations on their ability to recommend particular items based on the information they possess.

One way to alleviate coverage problems is by falling back to a baseline recommender when the collaborative filter cannot generate a prediction. Hybridizing algorithms with different coverage profiles can also help improve coverage. If the composite recommenders are of significantly different structure, this method can increase the system's ability to recommend some items. Latent factor models, such as SVD and PLSA, can also provide good coverage because the latent factors can connect users to items that they would not be connected to just through item or user neighborhoods.

McLaughlin and Herlocker [95] proposed a modified precision metric to measure the ability of a recommender to recommend items which takes coverage into account, penalizing algorithms with low coverage.

---

<sup>1</sup>Recall can rarely be directly measured. Data sets only contain rated items, while recall pertains to finding all relevant items regardless of whether they are already known to the user.

They then showed that a modified user–user recommender that operates by propagating belief distributions over rating offsets between similar users is able to achieve much better coverage at the expense of some accuracy compared to standard user–user and item–item CF. The balance with regards to coverage, accuracy, and computational requirements will vary by domain and application.

In addition to the impact of coverage on user needs, the system provider may also have an interest in coverage — if there are items which cannot be recommended, then it may be more difficult to sell those products.

#### 5.4.2 Robustness

In order for recommendations to be useful to users, they need to be robust and fair. Recommendations should reflect the true opinions of legitimate users of the site, with minimal degradation due to natural noise and free from bias introduced by malicious users attempting to manipulate the system’s ratings.

Having considered naturally-occurring noise in *Dealing with Noise*, we turn here to considerations for malicious users. An unscrupulous manufacturer desiring to sell more of their product may attempt to create accounts on e-commerce sites using recommender systems and strategically rating their product highly in order to cause it to be recommended to more users. Similarly, other groups may want to destroy a particular product’s recommendation level or otherwise manipulate the recommendation system’s output. Examples of this are so-called “Google Bombs”, where link profiles are set up to cause a particular page to rate highly for specific searches, such as “miserable failure” returning the White House page for Bush’s administration as the first result. Similar attacks have been carried out against Amazon.com, manipulating the “products you might also like” lists — a group of attackers succeeded in getting a book on gay sex listed as a related item for a Pat Robertson book.

Attacks based on false profiles, sometimes called *shilling* or *sybil* attacks, have received a fair amount of treatment in the literature. O’Mahony et al. [108] have examined the robustness of user-user

collaborative filtering to such attacks. Lam and Riedl [82] found that profile-based attacks can influence predictions and recommendation, but that item-item CF is less vulnerable than user-user to such attacks. Matrix factorization approaches (specifically PLSA) are also fairly robust to attack [103]. Further, it is difficult to reliably detect the presence of malicious user profiles using standard recommender accuracy metrics, although classifier-based methods can detect some errant profiles [102]. Resnick and Sami [120] proposed a reputation-based system, the *influence limiter*, which can be added to any collaborative filtering system and uses reputation scores, computed by determining which users contribute to accurate prediction of a target user's ratings, to limit the influence of individual users on the target user's recommendations.

Many currently known attacks exploit knowledge of the recommender algorithm in use, and sometimes additional information such as the average rating for a given item. Sometimes specific weaknesses can cause a marked increase in attack success. For example, recommenders based on Pearson correlation without similarity damping are susceptible to shilling attacks with small attack profiles, as two users with few ratings in common have abnormally high Pearson correlations [108].

So far, most research on the robustness of collaborative filtering to manipulation has focused on a few specific attack methods, most notably the creation of false profiles with the goal of artificially boosting or depressing an item in recommendation lists. Little is known about other attack methods or attacks aimed at different goals (such as an attack attempting to discount the credibility of the system by causing it to generate spurious or bogus recommendations). Resnick and Sami [121] have shown that there is an inherent tradeoff between manipulation-resistance and taking advantage of the information in user ratings; in order to resist manipulation by an attacker capable of creating an unbounded number of shill profiles, a system must discard all user ratings.

#### 5.4.3 Privacy

Recommendation is not the only need users have with respect to their relationship with a recommender system. Among other things, they

may also desire privacy: ratings data can communicate a fair amount about a user's preferences and habits, and users may want to restrict the extent to which that data can be discovered by other users in a system (or even the system itself). Further, privacy is a significant consideration when releasing data sets to the public. There has been notable success in taking anonymized data sets, such as the Netflix Prize set, and de-anonymizing them (determining the identities of particular users) [41, 105].

An immediate privacy concern, at least within the e-commerce domain, is the ability for users of a system to gain information about other users. Ramakrishnan et al. [116] analyzed this problem, finding that users who tastes straddle seemingly-disparate regions of the item space can be susceptible to identification by other users of the system with access to aggregate sales data. They advised refraining from using connections involving a small number of people in producing recommendations so that the observable behavior of the recommender is based on sufficiently large samples that it is infeasible to identify individual users. The requirements for this attack are fairly sophisticated, but feasible with vendors sharing data and utilizing third-party personalization services.

Research has also considered recommendation schemes that allow users to receive recommendations and/or predictions without any central party knowing all users' ratings. Canny [26, 27] proposed factor analysis with this goal in mind, using homomorphic encryption and zero-knowledge protocols to allow users to collaboratively compute a recommendation model without disclosing their individual ratings. Other matrix factorization techniques such as singular value decomposition have also been deployed to implement decentralized recommenders: the PocketLens algorithm used SVD with encryption to perform decentralized collaborative filtering on hand-held devices [101] and Polat and Du [113] proposed using SVD with randomly-perturbed data to preserve privacy.

Hybrid methods, involving central servers that are partially trusted, have also been considered. Notable among these is Alambic, an architecture which separates purchasing and recommendation [8]. It calls for recommendation providers to be distinct from e-commerce providers,

so that the e-commerce vendor does not know a user's history and the recommendation provider does not know their current activity. There have also been proposals to have users share and aggregate their ratings before submitting them to the central server [138].

Kobsa [75] provide a thorough analysis of the privacy concerns in personalization systems in general, including legally imposed conditions. We refer you to that work for further reading on this aspect of the user–recommender relationship.

## 5.5 Summary

Within results of equivalent accuracy or numerical quality, there is still sufficient variation that algorithms with equivalent accuracy performance can produce recommendations that are qualitatively different from each other in terms of user response. This provides opportunity to tailor a recommender system deployment to meet the particular needs of its users. This can be done by tuning the algorithm to the expected needs [99], providing (or automatically adjusting) parameters to influence its behavior [156], or switching algorithms based on the user's current task [156].

As with user interface design (and, indeed, design for any system that interacts with humans), clearly identifying the user tasks the system is intended support and what users expect from the system provides a useful framework for the design and evaluation of a recommender system. At the end of the day, users are more concerned with having their needs met or otherwise getting value from a system than numeric measures of its accuracy.

# 6

---

## User Experience

---

In order to move from predicting preference to meeting user needs, recommender systems must interact with their users in some fashion. They need to collect ratings, frequently requiring user interaction (although this interaction is occasionally indirect), and they must present the recommendations or predictions back to the user.

Recommenders also have the opportunity to integrate more broadly with user experience, particularly the social context in which recommendations are received and used. There have also been projects to use recommenders to affect user behavior in ways beyond purchasing products or seeking entertainment.

This section surveys a variety of work that has been done on user interaction with recommenders and on the ways in which recommenders interact with user experience and social contexts. Better understanding of human–recommender interaction and how recommenders affect and are affected by the people who use them is also an important direction for further recommender systems research.

### 6.1 Soliciting Ratings

As discussed in *Rating Scales*, a variety of methods have been tried for soliciting ratings from users of recommender systems. GroupLens

augmented standard newsreaders with buttons or commands for applying ratings of 1–5 stars [119]. Jester provided a continuous bar that recorded the position at which users clicked it as their rating (using an HTML image map) [50]. MovieLens uses drop-down menus or images for users to select 1–5 star (with half-star) ratings; some early recommenders interacted with users via e-mail.

The most common interface construct for allowing users to provide ratings is a rating widget of some kind displayed with the item either on an item info details or in its entry in a list. In general, the rating interface will need to integrate well with the rest of the application or site’s user experience, and standard human–computer interaction design principles apply (including using interface devices familiar to users from other systems). There is currently a good deal of inconsistency, even within a single system, with regards to how ratings are collected; Karvonen et al. [70] provide a heuristic evaluation of several popular web sites with rating or reputation interfaces and demonstrates a number of consistency issues. Live deployments are not all bad, however; Netflix star ratings and Pandora’s thumbs up/down are good examples of current best practices in rating interfaces.

Users prefer more granularity in their ratings interfaces — on a 5-star scale, they like to be able to give half-star ratings — so it is beneficial for the user experience to allow a relatively fine-grained rating process, but the increased granularity of ratings will not necessarily translate into more accurate recommendations [34].

## 6.2 Presenting Recommendations

Once the system has collected user ratings and computed recommendations, it must communicate those recommendations to the user so that they can evaluate and potentially act on the recommended items. Communicating predictions is fairly straightforward — in an item listing or details view, the predicted preference can be shown as a feature of the item. MovieLens does this, replacing the predicted preference with the user’s actual rating when it is an item they have rated.

Methods of presenting recommendations have changed as communications technology evolved. In the early 1990s, when the Web was

young and had not yet achieved widespread usage, recommenders such as BellCore's video recommender interacted with the user via e-mail: users e-mailed ratings of movies to `videos@bellcore.com` and received recommendations in an e-mail reply [62]. The majority of recommender systems are currently built into web sites. TV set-top boxes are also used as a delivery mechanism for recommendations [4], soliciting ratings via the remote control and presenting the user with a list of recommended shows with their air dates. Increasing usage of sophisticated cell phones have made mobile recommendations increasingly a reality [100]. Delivering recommendations via devices other than traditional PCs has the potential to bring recommendation much closer to the actual context of use for a wider variety of items — TV shows or movies can be immediately watched, and users can receive product recommendations while in the store or at the theater.

There is more diversity in approaches to showing recommendations. Some systems, such as MovieLens, merely have a view where items are displayed in a search results fashion ordered by predicted preference. Others display various forms of recommendations in-line with the browsing experience: many pages on Amazon.com, such as product information pages and shopping baskets, show recommendations based on the user's past history or the actions taken by other users with respect to the currently displayed item.

In environments where the cost of consumption is low and recommendation process is ongoing, the system can simply provide the user with the recommended item. This is done by Jester, where the system shows the user jokes and asks for ratings until the user decides to leave [50]. Music recommenders such as Pandora also frequently do this — Pandora starts playing the next song, and users can skip or thumb-down the song if they don't like it.

One specific aspect of communicating recommendations to users is that of explaining why particular items were recommended. Recommender systems are frequently “black boxes”, presenting recommendations to the user without any explanation of why the user might like the recommended item. Explanations seem to be useful, however, for helping the user be confident in the provided recommendations and to make better decisions based on them [60].

Explanation methods can stem from a variety of goals. Tintarev [144] identifies seven potential goals for explanations of recommendations: transparency of the recommender algorithm, scrutability of the results, trustworthiness of the recommendations, effectiveness for supporting decisions, persuasiveness, efficiency of decision-making, and satisfaction with the recommendation process.

User modelling work by Cook and Kay [32] demonstrated a means for showing users the system's model of them and showed that many users were interested in this information. The earliest systematic evaluation of recommender explanations was by Herlocker et al. [60]. While it did not show a change in users' enjoyment of recommended items based on explanations, that study did document a significant user interest in explanations of the recommendations they received. Bilgic and Mooney [17] argue that for many domains, user satisfaction with the item is more important than whether they act on the recommendation: satisfaction and effectiveness are more critical to evaluate than persuasiveness. A later focus group study suggested that explanations can help with user satisfaction by helping them better know what to expect [145].

For other stakeholders the balance of explanatory goals can be different. An e-commerce site typically uses recommendations to increase sales volume, increasing the importance of persuasion as a goal. This has limits, however — if the system has a reputation for recommending bad items it can lose trust with users and suffer in the long run.

Some work has been done on assessing the impact of the recommender interface on user reception of recommendations [54, 60]. Ultimately, however, this will likely have a good number of domain- and system-specific attributes. Further, it is an example of one of the aspects of recommender evaluation that cannot be done off-line: to measure user response to recommendation, it is necessary to test a system with actual users.

### **6.3 Recommending in Conversation**

Many recommender systems simply provide the user with a set of recommendations or predictions, and the only feedback mechanism provided to the user for refining future recommendations is the ability

to rate (or purchase) more items. *Conversational recommenders* seek to expand that, soliciting the user’s input to more accurately meet their needs through an ongoing cycle of recommendation and feedback (a *conversation*). In some ways, the resulting user experience is similar to that of Grundy [123], in which the user could reject the suggested book and receive a new suggestion. Conversational recommendation has predominantly been explored in the context of case-based reasoning systems [3, 46, 94] where recommendation is of a more content-based than collaborative nature.

McGinty and Smyth [94] divide the kinds of feedback which are solicited by various recommenders into four types: *value elicitation*, where the user provides a specific value for some feature as a hard constraint (e.g., “built by Sony”); *ratings*, where the user provides ratings for items; *critique*, where the user provides a specific correction to the system’s recommendation (e.g., “like this, but with more megapixels”); and *preference*, where the user selects the most desirable of the recommended items and the system replies with more items similar to the selected item.<sup>1</sup>

## 6.4 Recommending in Social Context

So far, the recommendation systems considered have all treated the user as a single, independent entity who will consume the recommendation without interaction with others. There has been work, however, on making recommender systems aware of social contexts and connections between users.

### 6.4.1 Group Recommenders

Group recommenders have the goal of recommending items to be experienced by a group of people, aiming to find an item (such as a movie or travel destination) which all members will enjoy [92]. PolyLens provided movie recommendations to groups by using MovieLens’s user–user collaborative filter to generate predictions for each user, producing the

---

<sup>1</sup>Under this taxonomy, traditional ratings-based recommender systems become a special case of conversational recommenders without an explicitly articulated recommend-refine cycle.

group's recommendations by merging the individual predictions using the "principle of least misery": the predicted preference for a movie was the lowest preference predicted for any group member [107]. Users enjoyed having the group recommendations, although there are privacy tradeoffs as some information about users' preferences can be inferred by their fellow group members.

Jameson [66] describes a number of challenges in building group recommenders. First is preference collection: while PolyLens simply used the ratings each user had previously provided, in some domains it can be desirable to allow users to see each other's stated preferences and/or interact with each other around the preference-stating process. Second, the preferences need to be aggregated and the recommendations produced. With user–user CF algorithms, this can be done either by blending individual recommendation lists for each user or by creating a composite pseudo-user for the group and generating recommendations for that user [107]. In either of these cases, it is necessary to have some aggregation function for combining preferences. O'Connor considered several methods for PolyLens, including minimum preference, maximum preference, and average preference; Jameson points out that it is important for this preference function to be *non-manipulable* so that no user can manipulate the group's recommendations by falsely stating their preferences. For his Travel Decision Forum system, he selected median as the aggregation function. This requirement is important, as manipulability has been a problem for deployed group recommenders. The initial trial of the MusicFX system for selecting fitness center background music allowed an individual to force a music change by giving the current music the minimum rating [92].

After the system has produced a set of recommendations for the group, it needs to provide the group with those recommendations and, in some cases, support the group as they finalize their decision. Jameson spends some time on the nuances of facilitating group decision around the resulting recommendation lists.

#### 6.4.2 Incorporating Trust in Recommenders

Even when recommendations are intended for individual users, users may have relationships with other users that can be used as an

additional source of information in the recommendation process. This is particularly the case with the rise of social networking, with extensive data available about peoples' connections to each other.

Various authors have proposed methods for integrating information about user's connections into recommender systems in the form of "trust-based recommenders". Massa and Avesani proposed a method for using users' statements of their trust in other users' opinions to weight user ratings by estimated trust rather than similarity in user-user CF when producing predictions and recommendations and built a ski mountaineering site around it [12, 91]. Golbeck [47] used a similar integration method, based on a different trust estimation algorithm, for movie recommendation.

Trust is a complicated concept, and representing and estimating it computationally is difficult. While many systems have a good deal of information about peoples' connections, how those connections correspond to trust in recommendation is not obvious. Even annotating the network with trust ratings is problematic, with confidence and notions such as active distrust making a useful single "estimated trust" value an elusive goal. There is continued work, however, on various methods for estimating and propagating trust through social networks [12, 47, 48, 114, 155]. Ziegler and Golbeck [154] found that trust and conventional rating similarities are correlated. Guy et al. also found that users tended to find recommendations of web sites, discussion forums, and other social software artifacts more interesting when they were recommended from the user's social connections rather than users with similar preference histories [54].

#### 6.4.3 Recommending Social Connections

Some recommender projects have gone beyond recognizing that recommendations are used in social contexts by using recommenders to alter or enhance the structure of that social context — recommending social connections or friends.

Kautz et al. [72] did early work in this direction, building the Referral Web system that allowed users to query their social network;

McDonald [93] extended these ideas by building a recommender for individuals with particular expertise from the user's social network or organization. These systems enabled users to find people in their social network or organization that could provide them with particular expertise or otherwise meet a need they had.

Terveen and McDonald [143] provide a survey and overview of social connection recommendation, covering information-need-based recommendation such as that of Kautz et al. and McDonald as well as other applications of social connection recommendation. Recent work has focused on recommending connections in the contexts of online social networks [28].

## 6.5 Shaping User Experience with Recommendations

The interaction between recommender systems and user experience goes beyond providing a useful user experience around recommendations. Recommendations can also be used to alter user experience and behavior. Cosley et al. [34] found that predictions influence user rating behavior, but there is interest in using recommender systems to influence user behavior in broader contexts.

SuggestBot, an implementation of *intelligent task routing*, uses recommender algorithms to suggest pages that Wikipedia editors may want to edit [33]. Intelligent task routing aims to help people do work on community-maintained projects by guiding them to work that needs to be done and which matches with their interests as inferred from prior work history.

In online social networking tools, recommenders have been successfully deployed to improve user experience and engagement. Recommending connections to existing users of IBM's internal social networking site SocialBlue (formerly known as BeeHive) resulted in users expanding their connections. Users' social network information was more useful for finding people they already knew on the system, but recommender algorithms based on similarity in user profile contents were more effective at helping users discover new friends [28]. Recommending

social connections and profile content to new users resulted in the users being more active on the site [42].

Recommenders therefore have the potential not just to meet information needs, but to shape and guide user behavior both on-line and in the real world.

# 7

---

## Conclusion and Resources

---

Recommender systems have become ubiquitous. People use them to find books, music, news, smart phones, vacation trips, and romantic partners. Nearly every product, service, or type of information has recommenders to help people select from among the myriad alternatives the few they would most appreciate. Sustaining these commercial applications is a vibrant research community, with creative interaction ideas, powerful new algorithms, and careful experiments. Still, there are many challenges for the field, especially at the interaction between research and commercial practice. We have framed these challenges here in a five part taxonomy.

**Algorithms:** Recommender systems researchers have developed a suite of highly effective algorithms for the basic problem of recommending a set of substitutable goods from a large population of similar goods to individual users. There are however many remaining algorithmic challenges, most involving richer sets of data about the users, the items, the interactions between the users and the items, or the relationships among groups of users or groups of items. For instance, how can such sources

as Amazon reviews and Twitter posts about items be incorporated into recommendations. We expect to see a wide variety of approaches, but are particularly optimistic about algorithms that generalize to process multiple parallel matrices of user and item data simultaneously.

**Data:** The data used by recommender systems are sometimes biased in unexpected ways that can have a dramatic effect on outcomes. For instance, in designing algorithms for selecting items for new users to rate in MovieLens we found that the original MovieLens algorithm, which was performing poorly in practice, nearly always scored higher in offline tests than the newer algorithms we were designing. The reason is that since new users had been using the original algorithm for years, the dataset we were using for offline testing was much more likely to have a rating for items chosen by that algorithm than by any new algorithm. Similarly, nearly all rating data sets are strongly biased toward high ratings, because users are careful to only choose to consume items they suspect they will like. This class of problems is challenging to solve in general, but there are some elegant approaches emerging for specific instances. For instance, researchers have shown that datasets can be effectively statistically unbiased for some popularity biases [88].

**User experience:** The goal of recommender systems is to improve user experience. In both research and practice, crafting the user experience to fit the application and the user lifecycle remains a substantial challenge. One challenge is to adapt the nature of recommendations as the user gains more experience with the recommender — a new user may need more verifiable recommendations and may lack the trust needed for high-risk recommendations. A similar challenge is how to balance serving the individual now (with the best available recommendation) vs. serving the individual and community long-term (with recommendations that help the recommender system learn). These challenges vary with different domains, and particularly with different usage patterns. Part of the research challenge is to design interfaces that give users control over

the recommendation process without overwhelming the user or rendering the tool too complicated for novice users.

**Evaluation and metrics:** Evaluation of recommender systems has advanced significantly over the past decade, but many challenges remain. Even with a better understanding of the relationships among different algorithm performance metrics, the field struggles to standardize evaluation. As a result, too often research results are incomparable with prior published results, even when based on the same data sets. Examples of this problem include the lack of standard treatment of items for which the recommender is unable to make a prediction. The broader goal of user-centered holistic evaluation, including A/B testing of the short- and long-term effects of recommendation differences, is still met by only a few research groups and companies that have the live systems and resources for such evaluation. Deploying innovative recommenders is still too hard, and there is a substantial need for research platforms where innovations can be tested without first building up a community of thousands of users.

**Social impact:** Because collaborative filtering recommender systems must by their nature collect substantial personalized data about their users, they face important privacy and security challenges. Researchers have been approaching these challenges head-on by attempting to develop ways to collect and store data in such a way that extracting personalized data is provably difficult. Recommender systems also have some unique community challenges. One such challenge is about the relationship between the recommender system and its community of users. How do the users know that the recommender system is providing honest recommendations? Furthermore, the existence of the recommender may influence the structure of the community over time. If users are choosing items to read based on personalized recommendations, over time they may cluster into groups of like-minded individuals, *balkanizing* the community into groups who seldom interact with people with whom they do not agree. We hope in the future to see recommender systems

that explicitly react to combat this type of damage to the community structure, ensuring that not only individuals, but also the surrounding community benefit from the existence of the recommender.

## 7.1 Resources

Given the broad application of recommenders and the substantial set of open problems, the field of recommender systems research promises to remain active for years to come. We conclude this survey with a brief review of resources we feel would be useful for both academic and industrial explorations of recommender systems.

**Publication venues:** Recommender systems research continues to progress rapidly, with many papers appearing each year in venues including: ACM Recommender Systems (RecSys) User Modeling, Adaptation and Personalization (UMAP) ACM Conference on Human Factors in Computing Systems (CHI) ACM Conference on Computer-Supported Cooperative Work (CSCW) ACM Conference on Information Retrieval (SIGIR) ACM Conference on Knowledge Discovery and Data Mining (KDD) ACM Transactions on Information Systems User Modeling and User-Adapted Interaction and many other workshops, conferences, and journals.

**Software implementations:** There are a variety of collaborative filtering implementations available and ready for use in projects, many under open source licenses, including the following:

- LensKit by GroupLens Resesearch, Java.  
<http://lenskit.grouplens.org>
- MyMediaLite by MyMedia at University of Hildesheim, C#/NET.  
<http://www.ismll.uni-hildesheim.de/mymedialite/>
- easyrec, Java with RESTful HTTP API. <http://easyrec.org>

- Mahout Taste by the Apache project, Java.  
<http://mahout.apache.org/>
- Cofi by Daniel Lemire et al., Java. <http://www.nongnu.org/cofi/>
- SUGGEST by George Karypis, binary-only, C API.  
[http://glaros.dtc.umn.edu/gkhome/suggest/  
download](http://glaros.dtc.umn.edu/gkhome/suggest/download)

**Data sets:** For evaluating and tuning recommender performance, commonly-used, readily-available data sets include the following:

- MovieLens: movie ratings (100K, 1M, and 10M ratings sets, the latter includes 100K tag applications). With the availability of larger data sets, we do not recommend using the 100K for other than development and preliminary testing; larger data sets will provide more robust validation of algorithms.  
<http://www.grouplens.org/node/73>
- Jester: ratings of jokes.  
<http://eigentaste.berkeley.edu/dataset/>
- BookCrossing: book ratings with user demographic information.  
[http://www.informatik.uni-freiburg.de/  
~cziegler/BX/](http://www.informatik.uni-freiburg.de/~cziegler/BX/)
- WebScope — Yahoo! provides a number of web user behavior data sets from various Yahoo! services through their WebScope program.  
<http://webscope.sandbox.yahoo.com>
- RecLab — RichRelevance provides a recommendation evaluation framework and simulated data set for evaluating recommender performance for e-commerce.  
[http://code.richrelevance.com/reclab.](http://code.richrelevance.com/reclab)

---

Work on this survey was funded by the National Science Foundation under grant IIS 05-34939.

## References

---

- [1] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.
- [2] E. Agichtein, E. Brill, S. Dumais, and R. Ragno, “Learning user interaction models for predicting web search result preferences,” in *ACM SIGIR '06*, pp. 3–10, ACM, 2006.
- [3] D. Aha and L. Breslow, “Refining conversational case libraries,” in *Case-Based Reasoning Research and Development*, vol. 1266 of *Lecture Notes in Computer Science*, pp. 267–278, Springer, 1997.
- [4] K. Ali and W. van Stam, “TiVo: Making show recommendations using a distributed collaborative filtering architecture,” in *ACM KDD '04*, pp. 394–401, ACM, 2004.
- [5] X. Amatriain, J. Pujol, and N. Oliver, “I like it... I like it not: Evaluating user ratings noise in recommender systems,” in *UMAP 2009*, vol. 5535 of *Lecture Notes in Computer Science*, pp. 247–258, Springer, 2009.
- [6] X. Amatriain, J. M. Pujol, N. Tintarev, and N. Oliver, “Rate it again: Increasing recommendation accuracy by user re-rating,” in *ACM RecSys '09*, pp. 173–180, ACM, 2009.
- [7] Amazon.com, “Q4 2009 Financial Results,” Earnings Report Q4-2009, January 2010.
- [8] E. Aïmeur, G. Brassard, J. Fernandez, and F. M. Onana, “Alambic: A privacy-preserving recommender system for electronic commerce,” *International Journal of Information Security*, vol. 7, no. 5, pp. 307–334, October 2008.

- [9] T. Amoo and H. H. Friedman, "Do numeric values influence subjects' responses to rating scales?," *Journal of International Marketing and Marketing Research*, vol. 26, pp. 41–46, February 2001.
- [10] A. Ansari, S. Essegaeier, and R. Kohli, "Internet recommendation systems," *Journal of Marketing Research*, vol. 37, no. 3, pp. 363–375, August 2000.
- [11] C. Avery and R. Zeckhauser, "Recommender systems for evaluating computer messages," *Communications of the ACM*, vol. 40, no. 3, pp. 88–89, ACM ID: 245127, March 1997.
- [12] P. Avesani, P. Massa, and R. Tiella, "A trust-enhanced recommender system application: Moleskiing," in *ACM SAC '05*, pp. 1589–1593, ACM, 2005.
- [13] M. Balabanović and Y. Shoham, "Fab: Content-based, collaborative recommendation," *Communications of the ACM*, vol. 40, no. 3, pp. 66–72, 1997.
- [14] R. M. Bell and Y. Koren, "Scalable collaborative filtering with jointly derived neighborhood interpolation weights," in *IEEE ICDM 2007*, pp. 43–52, 2007.
- [15] J. Bennett and S. Lanning, "The netflix prize," in *KDD Cup and Workshop '07*, 2007.
- [16] M. W. Berry, S. T. Dumais, and G. W. O'Brien, "Using linear algebra for intelligent information retrieval," *SIAM Review*, vol. 37, no. 4, pp. 573–595, December 1995.
- [17] M. Bilgic and R. J. Mooney, "Explaining recommendations: Satisfaction vs. promotion," in *Beyond Personalization 2005: A Workshop on the Next Stage of Recommender Systems Research*, pp. 13–18, 2005.
- [18] D. Billsus and M. J. Pazzani, "Learning collaborative information filters," in *AAAI 2008 Workshop on Recommender Systems*, 1998.
- [19] D. Billsus and M. J. Pazzani, "A personal news agent that talks, learns and explains," in *AGENTS '99*, pp. 268–275, ACM, 1999.
- [20] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, March 2003.
- [21] M. Brand, "Fast online svd revisions for lightweight recommender systems," in *SIAM International Conference on Data Mining*, pp. 37–46, SIAM, 2003.
- [22] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *UAI 1998*, pp. 43–52, AAAI, 1998.
- [23] P. Brusilovsky, "Methods and techniques of adaptive hypermedia," *User Modeling and User-Adapted Interaction*, vol. 6, no. 2, pp. 87–129, July 1996.
- [24] R. Burke, "Hybrid recommender systems: Survey and experiments," *User Modeling and User-Adapted Interaction*, vol. 12, no. 4, pp. 331–370, November 2002.
- [25] R. Burke, "Evaluating the dynamic properties of recommendation algorithms," in *ACM RecSys '10*, pp. 225–228, ACM, 2010.
- [26] J. Canny, "Collaborative filtering with privacy," in *IEEE Symposium on Security and Privacy 2002*, pp. 45–57, IEEE Computer Society, 2002.
- [27] J. Canny, "Collaborative filtering with privacy via factor analysis," in *ACM SIGIR '02*, pp. 238–245, ACM, 2002.
- [28] J. Chen, W. Geyer, C. Dugan, M. Muller, and I. Guy, "Make new friends, but keep the old: Recommending people on social networking sites," in *ACM CHI '09*, pp. 201–210, ACM, 2009.

- [29] L. Chen and P. Pu, "Evaluating critiquing-based recommender agents," in *AAAI 2006*, vol. 21, pp. 157–162, AAAI, 2006.
- [30] Y. H. Chien and E. I. George, "A bayesian model for collaborative filtering," in *7th International Workshop on Artificial Intelligence and Statistics*, 1999.
- [31] B. H. Clark, "Marketing performance measures: History and interrelationships," *Journal of Marketing Management*, vol. 15, no. 8, pp. 711–732, November 1999.
- [32] R. Cook and J. Kay, "The justified user model: A viewable, explained user model," in *User Modelling 1994*, 1994.
- [33] D. Cosley, D. Frankowski, L. Terveen, and J. Riedl, "SuggestBot: using intelligent task routing to help people find work in wikipedia," in *ACM IUI '07*, pp. 32–41, ACM, 2007.
- [34] D. Cosley, S. K. Lam, I. Albert, J. A. Konstan, and J. Riedl, "Is seeing believing?: How recommender system interfaces affect users' opinions," in *ACM CHI '03*, pp. 585–592, ACM, 2003.
- [35] B. Dahlen, J. Konstan, J. Herlocker, N. Good, A. Borchers, and J. Riedl, "Jump-starting MovieLens: User benefits of starting a collaborative filtering system with 'dead data,'" Technical Report 98-017, University of Minnesota, Minneapolis, MN, March, 1998.
- [36] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.
- [37] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, ArticleType: research-article/Full publication date: 1977/Copyright © 1977 Royal Statistical Society, Janaury 1977.
- [38] M. Deshpande and G. Karypis, "Item-based top-N recommendation algorithms," *ACM Transactions on Information Systems*, vol. 22, no. 1, pp. 143–177, 2004.
- [39] M. D. Ekstrand, P. Kannan, J. A. Stemper, J. T. Butler, J. A. Konstan, and J. T. Riedl, "Automatically building research reading lists," in *ACM RecSys '10*, pp. 159–166, ACM, 2010.
- [40] B. Fields, C. Rhodes, and M. d'Inverno, "Using song social tags and topic models to describe and compare playlists," in *Workshop on Music Recommendation and Discovery 2010*, 633, CEUR, September 2010.
- [41] D. Frankowski, D. Cosley, S. Sen, L. Terveen, and J. Riedl, "You are what you say: Privacy risks of public mentions," in *ACM SIGIR '06*, pp. 565–572, ACM, 2006.
- [42] J. Freyne, M. Jacovi, I. Guy, and W. Geyer, "Increasing engagement through early recommender intervention," in *ACM RecSys '09*, pp. 85–92, ACM, 2009.
- [43] H. H. Friedman and T. Amoo, "Rating the rating scales," *Journal of Marketing Management*, vol. 9, no. 3, pp. 114–123, 1999.
- [44] S. Funk, "Netflix update: Try this at home," <http://sifter.org/~simon/journal/20061211.html>, Archived by WebCite at <http://www.webcitation.org/5pVQphxrD>, December 2006.

- [45] R. Garland, “The mid-point on a rating scale: Is it desirable?,” *Marketing Bulletin*, vol. 2, pp. 66–70, May 1991.
- [46] M. Göker and C. Thompson, “Personalized conversational case-based recommendation,” in *Advances in Case-Based Reasoning*, vol. 1898 of *Lecture Notes in Computer Science*, pp. 29–82, Springer, 2000.
- [47] J. Golbeck, “Generating predictive movie recommendations from trust in social networks,” in *International Conference on Trust Management*, vol. 3986 of *Lecture Notes in Computer Science*, pp. 93–104, Springer, 2006.
- [48] J. Golbeck, “Trust on the World Wide Web: A survey,” *Foundations and Trends® in Web Science*, vol. 1, no. 2, pp. 131–197, 2006.
- [49] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, “Using collaborative filtering to weave an information tapestry,” *Communications of the ACM*, vol. 35, no. 12, pp. 61–70, 1992.
- [50] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, “Eigentaste: A constant time collaborative filtering algorithm,” *Information Retrieval*, vol. 4, no. 2, pp. 133–151, July 2001.
- [51] G. Gorrell, “Generalized Hebbian algorithm for incremental singular value decomposition in natural language processing,” in *EACL 2006*, pp. 97–104, ACL, 2006.
- [52] M. Grigoriev, “Intelligent multimedia management system,” 2003.
- [53] A. Gunawardana and G. Shani, “A survey of accuracy evaluation metrics of recommendation tasks,” *Journal of Machine Learning Research*, vol. 10, pp. 2935–2962, 2009.
- [54] I. Guy, N. Zwerdling, D. Carmel, I. Ronen, E. Uziel, S. Yoge, and S. Ofek-Koifman, “Personalized recommendation of social software items based on social relations,” in *ACM RecSys '09*, pp. 53–60, ACM, 2009.
- [55] D. L. Hansen and J. Golbeck, “Mixing it up: Recommending collections of items,” in *ACM CHI '09*, pp. 1217–1226, ACM, 2009.
- [56] F. M. Harper, X. Li, Y. Chen, and J. A. Konstan, “An economic model of user rating in an online recommender system,” in *User Modeling 2005*, vol. 3538 of *Lecture Notes in Computer Science*, pp. 307–316, Springer, August 2005.
- [57] D. A. Harrison and K. J. Klein, “What’s the difference? Diversity constructs as separation, variety, or disparity in organizations,” *Academy of Management Review*, vol. 32, no. 4, pp. 1199–1228, Oct 2007.
- [58] J. Herlocker, J. A. Konstan, and J. Riedl, “An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms,” *Information Retrieval*, vol. 5, no. 4, pp. 287–310, 2002.
- [59] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, “An algorithmic framework for performing collaborative filtering,” in *ACM SIGIR '99*, pp. 230–237, ACM, 1999.
- [60] J. L. Herlocker, J. A. Konstan, and J. Riedl, “Explaining collaborative filtering recommendations,” in *ACM CSCW '00*, pp. 241–250, ACM, 2000.
- [61] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, “Evaluating collaborative filtering recommender systems,” *ACM Transactions on Information Systems*, vol. 22, no. 1, pp. 5–53, 2004.

- [62] W. Hill, L. Stead, M. Rosenstein, and G. Furnas, “Recommending and evaluating choices in a virtual community of use,” in *ACM CHI ’95*, pp. 194–201, ACM Press/Addison-Wesley Publishing Co., 1995.
- [63] W. Hill and L. Terveen, “Using frequency-of-mention in public conversations for social filtering,” in *ACM CSCW ’96*, pp. 106–112, ACM, 1996.
- [64] T. Hofmann, “Probabilistic latent semantic indexing,” in *ACM SIGIR ’99*, pp. 50–57, ACM, 1999.
- [65] T. Hofmann, “Latent semantic models for collaborative filtering,” *ACM Transactions on Information Systems*, vol. 22, no. 1, pp. 89–115, 2004.
- [66] A. Jameson, “More than the sum of its members: Challenges for group recommender systems,” in *Working Conference on Advanced Visual Interfaces*, pp. 48–54, ACM, 2004.
- [67] X. Jin, Y. Zhou, and B. Mobasher, “Web usage mining based on probabilistic latent semantic analysis,” in *ACM KDD ’04*, pp. 197–205, ACM, 2004.
- [68] D. Kahneman, P. P. Wakker, and R. Sarin, “Back to bentham? Explorations of experienced utility,” *The Quarterly Journal of Economics*, vol. 112, no. 2, pp. 375–405, ArticleType: research-article/Issue Title: In Memory of Amos Tversky (1937–1996)/Full publication date: May, 1997/Copyright© 1997 The MIT Press, May 1997.
- [69] J. Karlgren, “Newsgroup clustering based on user behavior — a recommendation algebra,” Technical Report, European Research Consortium for Informatics and Mathematics at SICS, 1994.
- [70] K. Karvonen, S. Shibasaki, S. Nunes, P. Kaur, and O. Immonen, “Visual nudges for enhancing the use and produce of reputation information,” in *Workshop on User-Centric Evaluation of Recommender Systems and Their Interfaces*, September 2010.
- [71] G. Karypis, “Evaluation of item-based top-N recommendation algorithms,” in *ACM CIKM ’01*, pp. 247–254, ACM, 2001.
- [72] H. Kautz, B. Selman, and M. Shah, “Referral Web: Combining social networks and collaborative filtering,” *Communications of the ACM*, vol. 40, no. 3, pp. 63–65, 1997.
- [73] B. Kitts, D. Freed, and M. Vrieze, “Cross-sell: A fast promotion-tunable customer-item recommendation method based on conditionally independent probabilities,” in *ACM KDD ’00*, pp. 437–446, ACM, 2000.
- [74] B. P. Knijnenburg, L. Schmidt-Thieme, and D. G. Bollen, “Workshop on user-centric evaluation of recommender systems and their interfaces,” in *ACM RecSys ’10*, p. 383, ACM, 2010.
- [75] A. Kobsa, “Privacy-enhanced web personalization,” in *The Adaptive Web*, pp. 628–670, Springer, 2007.
- [76] R. Kohavi, R. M. Henne, and D. Sommerfield, “Practical guide to controlled experiments on the web: Listen to your customers not to the HiPPO,” in *ACM KDD ’07*, pp. 959–967, ACM, ACM ID: 1281295, 2007.
- [77] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, “Controlled experiments on the web: Survey and practical guide,” *Data Mining and Knowledge Discovery*, vol. 18, no. 1, pp. 140–181, 2008.

- [78] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl, “GroupLens: applying collaborative filtering to Usenet news,” *Communications of the ACM*, vol. 40, no. 3, pp. 77–87, 1997.
- [79] Y. Koren, “Factorization meets the neighborhood: A multifaceted collaborative filtering model,” in *ACM KDD '08*, pp. 426–434, ACM, 2008.
- [80] Y. Koren, “Collaborative filtering with temporal dynamics,” *Communications of the ACM*, vol. 53, no. 4, pp. 89–97, 2010.
- [81] M. Kurucz, A. A. Benczúr, and K. Csalogány, “Methods for large scale SVD with missing values,” in *KDD Cup and Workshop 2007*, August 2007.
- [82] S. K. Lam and J. Riedl, “Shilling recommender systems for fun and profit,” in *ACM WWW '04*, pp. 393–402, ACM, 2004.
- [83] T. Landgrebe, P. Paclik, R. Duin, and A. Bradley, “Precision-recall operating characteristic (P-ROC) curves in imprecise environments,” in *ICPR 2006*, pp. 123–127, IEEE Computer Society, 2006.
- [84] N. Lathia, “Evaluating Collaborative Filtering Over Time,” PhD thesis, University College London, London, UK, June, 2010.
- [85] N. Lathia, S. Hailes, and L. Capra, “Evaluating collaborative filtering over time,” in *SIGIR 09 Workshop on the Future of IR Evaluation*, July 2009.
- [86] N. Lathia, S. Hailes, and L. Capra, “Temporal collaborative filtering with adaptive neighbourhoods,” in *ACM SIGIR '09*, pp. 796–797, ACM, 2009.
- [87] G. Linden, B. Smith, and J. York, “Amazon.com recommendations: Item-to-item collaborative filtering,” *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, 2003.
- [88] B. M. Marlin and R. S. Zemel, “Collaborative prediction and ranking with non-random missing data,” in *ACM RecSys '09*, pp. 5–12, ACM, 2009.
- [89] B. M. Marlin, R. S. Zemel, S. Roweis, and M. Slaney, “Collaborative filtering and the missing at random assumption,” in *UAI '07*, pp. 50–54, AUAI, 2007.
- [90] F. J. Martin, “RecSys '09 industrial keynote: Top 10 lessons learned developing deploying and operating real-world recommender systems,” in *ACM RecSys '09*, pp. 1–2, ACM, 2009.
- [91] P. Massa and P. Avesani, “Trust-Aware collaborative filtering for recommender systems,” in *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, vol. 3290 of *Lecture Notes in Computer Science*, pp. 275–301, Springer, 2004.
- [92] J. F. McCarthy and T. D. Anagnost, “MusicFX: an arbiter of group preferences for computer supported collaborative workouts,” in *ACM CSCW '98*, pp. 363–372, ACM, 1998.
- [93] D. W. McDonald, “Evaluating expertise recommendations,” in *ACM GROUP '01*, pp. 214–223, ACM, ACM ID: 500319, 2001.
- [94] L. McGinty and B. Smyth, “Adaptive selection: An analysis of critiquing and preference-based feedback in conversational recommender systems,” *International Journal of Electronic Commerce*, vol. 11, no. 2, pp. 35–57, 2006.
- [95] M. R. McLaughlin and J. L. Herlocker, “A collaborative filtering algorithm and evaluation metric that accurately model the user experience,” in *ACM SIGIR '04*, pp. 329–336, Sheffield, United Kingdom: ACM, 2004.

## 170 References

- [96] S. McNee, S. Lam, J. Konstan, and J. Riedl, “Interfaces for eliciting new user preferences in recommender systems,” in *User Modeling 2003*, vol. 2702, pp. 178–187, Springer, 2003.
- [97] S. M. McNee, I. Albert, D. Cosley, P. Gopalkrishnan, S. K. Lam, A. M. Rashid, J. A. Konstan, and J. Riedl, “On the recommending of citations for research papers,” in *ACM CSCW ’02*, pp. 116–125, ACM, 2002.
- [98] S. M. McNee, J. Riedl, and J. A. Konstan, “Being accurate is not enough: How accuracy metrics have hurt recommender systems,” in *ACM CHI ’06 Extended Abstracts*, pp. 1097–1101, ACM, 2006.
- [99] S. M. McNee, J. Riedl, and J. A. Konstan, “Making recommendations better: An analytic model for human-recommender interaction,” in *ACM CHI ’06 Extended Abstracts*, pp. 1103–1108, ACM, 2006.
- [100] B. N. Miller, I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl, “MovieLens unplugged: Experiences with an occasionally connected recommender system,” in *ACM IUI ’03*, pp. 263–266, ACM, 2003.
- [101] B. N. Miller, J. A. Konstan, and J. Riedl, “PocketLens: Toward a personal recommender system,” *ACM Transactions on Information Systems*, vol. 22, no. 3, pp. 437–476, 2004.
- [102] B. Mobasher, R. Burke, R. Bhaumik, and C. Williams, “Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness,” *ACM Transactions on Internet Technology*, vol. 7, no. 4, p. 23, 2007.
- [103] B. Mobasher, R. Burke, and J. Sandvig, “Model-based collaborative filtering as a defense against profile injection attacks,” in *AAAI 2006*, pp. 1388–1393, AAAI, 2006.
- [104] M. Morita and Y. Shinoda, “Information filtering based on user behavior analysis and best match text retrieval,” in *ACM SIGIR ’94*, pp. 272–281, Springer-Verlag, 1994.
- [105] A. Narayanan and V. Shmatikov, “Robust de-anonymization of large sparse datasets,” in *IEEE Symposium on Security and Privacy 2008*, pp. 111–125, IEEE Computer Society, 2008.
- [106] D. Oard and J. Kim, “Implicit feedback for recommender systems,” in *AAAI Workshop on Recommender Systems*, Madison, Wisconsin, 1998.
- [107] M. O’Connor, D. Cosley, J. A. Konstan, and J. Riedl, “PolyLens: a recommender system for groups of users,” in *ECSCW 2001*, pp. 199–218, Kluwer Academic Publishers, 2001.
- [108] M. O’Mahony, N. Hurley, N. Kushmerick, and G. Silvestre, “Collaborative recommendation: A robustness analysis,” *ACM Transactions on Internet Technology*, vol. 4, no. 4, pp. 344–377, 2004.
- [109] M. P. O’Mahony, N. J. Hurley, and G. C. Silvestre, “Detecting noise in recommender system databases,” in *ACM IUI ’06*, pp. 109–115, ACM, 2006.
- [110] A. Paterek, “Improving regularized singular value decomposition for collaborative filtering,” in *KDD Cup and Workshop 2007*, August 2007.
- [111] D. M. Pennock, E. Horvits, and C. L. Giles, “Social choice theory and recommender systems: Analysis of the axiomatic foundations of collaborative filtering,” in *AAAI 2000*, AAAI, 2000.

- [112] D. M. Pennock, E. Horvitz, S. Lawrence, and C. L. Giles, “Collaborative filtering by personality diagnosis: A hybrid memory-and model-based approach,” in *UAI 2000*, pp. 473–480, AUAI, 2000.
- [113] H. Polat and W. Du, “SVD-based collaborative filtering with privacy,” in *ACM SAC '05*, pp. 791–795, ACM, 2005.
- [114] A. Popescul, L. H. Ungar, D. M. Pennock, and S. Lawrence, “Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments,” in *UAI 2001*, pp. 437–444, Morgan Kaufmann Publishers Inc., 2001.
- [115] G. Potter, “Putting the collaborator back into collaborative filtering,” in *KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, pp. 1–4, ACM, 2008.
- [116] N. Ramakrishnan, B. Keller, B. Mirza, A. Grama, and G. Karypis, “Privacy risks in recommender systems,” *IEEE Internet Computing*, vol. 5, no. 6, pp. 54–63, 2001.
- [117] A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, and J. Riedl, “Getting to know you: Learning new user preferences in recommender systems,” in *ACM IUI '02*, pp. 127–134, ACM, 2002.
- [118] J. Reilly, J. Zhang, L. McGinty, P. Pu, and B. Smyth, “Evaluating compound critiquing recommenders: A real-user study,” in *ACM EC '07*, pp. 114–123, ACM, ACM ID: 1250929, 2007.
- [119] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, “Grouplens: an open architecture for collaborative filtering of netnews,” in *ACM CSCW '94*, pp. 175–186, ACM, 1994.
- [120] P. Resnick and R. Sami, “The influence limiter: Provably manipulation-resistant recommender systems,” in *ACM RecSys '07*, pp. 25–32, ACM, 2007.
- [121] P. Resnick and R. Sami, “The information cost of manipulation-resistance in recommender systems,” in *ACM RecSys '08*, pp. 147–154, ACM, 2008.
- [122] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, eds., *Recommender Systems Handbook*. Springer, 2010.
- [123] E. Rich, “User modeling via stereotypes,” *Cognitive Science*, vol. 3, no. 4, pp. 329–354, October 1979.
- [124] C. J. V. Rijsbergen, *Information Retrieval*. Butterworth-Heinemann, 1979.
- [125] R. Salakhutdinov, A. Mnih, and G. Hinton, “Restricted Boltzmann machines for collaborative filtering,” in *ACM ICML '07*, pp. 791–798, ACM, 2007.
- [126] G. Salton, “The state of retrieval system evaluation,” *Information Processing and Management*, vol. 28, no. 4, pp. 441–449, July 1992.
- [127] T. D. Sanger, “Optimal unsupervised learning in a single-layer linear feedforward neural network,” *Neural Networks*, vol. 2, no. 6, pp. 459–473, 1989.
- [128] B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Analysis of recommendation algorithms for e-commerce,” in *ACM EC '00*, pp. 158–167, ACM, ACM ID: 352887, 2000.
- [129] B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Incremental SVD-based algorithms for highly scaleable recommender systems,” in *ICCIT 2002*, 2002.
- [130] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Reidl, “Item-based collaborative filtering recommendation algorithms,” in *ACM WWW '01*, pp. 285–295, ACM, 2001.

## 172 References

- [131] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. T. Riedl, “Application of dimensionality reduction in recommender system — a case study,” in *WebKDD 2000*, 2000.
- [132] J. B. Schafer, J. A. Konstan, and J. Riedl, “E-Commerce recommendation applications,” *Data Mining and Knowledge Discovery*, vol. 5, no. 1, pp. 115–153, Janaury 2001.
- [133] A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock, “Methods and metrics for cold-start recommendations,” in *ACM SIGIR '02*, pp. 253–260, ACM, 2002.
- [134] S. Sen, F. M. Harper, A. LaPitz, and J. Riedl, “The quest for quality tags,” in *ACM GROUP '07*, pp. 361–370, ACM, 2007.
- [135] G. Shani and A. Gunawardana, “Evaluating recommendation systems,” in *Recommender Systems Handbook*, (F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, eds.), pp. 257–297, Springer, 2010.
- [136] G. Shani, D. Heckerman, and R. I. Brafman, “An MDP-based recommender system,” *Journal of Machine Learning Research*, vol. 6, pp. 1265–1295, 2005.
- [137] U. Shardanand and P. Maes, “Social information filtering: Algorithms for automating “word of mouth”,” in *ACM CHI '95*, pp. 210–217, ACM Press/Addison-Wesley Publishing Co., 1995.
- [138] R. Shokri, P. Pedarsani, G. Theodorakopoulos, and J. Hubaux, “Preserving privacy in collaborative filtering through distributed aggregation of offline profiles,” in *ACM RecSys '09*, pp. 157–164, ACM, 2009.
- [139] B. Smyth, “Case-based recommendation,” in *The Adaptive Web*, vol. 4321 of *Lecture Notes in Computer Science*, (P. Brusilovsky, A. Kobsa, and W. Nejdl, eds.), pp. 342–376, Springer, 2007.
- [140] X. Su and T. Khoshgoftaar, “A survey of collaborative filtering techniques,” *Advances in Artificial Intelligence*, vol. 2009, p. 19, August 2009.
- [141] K. Swearingen and R. Sinha, “Interaction design for recommender systems,” in *DIS 2002*, ACM, 2002.
- [142] J. A. Swets, “Information retrieval systems,” *Science*, vol. 141, no. 3577, pp. 245–250, July 1963.
- [143] L. Terveen and D. W. McDonald, “Social matching: A framework and research agenda,” *ACM Transactions on Computer-Human Interaction*, vol. 12, no. 3, pp. 401–434, ACM ID: 1096740, September 2005.
- [144] N. Tintarev, “Explanations of recommendations,” in *ACM RecSys '07*, pp. 203–206, ACM, 2007.
- [145] N. Tintarev and J. Masthoff, “Effective explanations of recommendations: User-centered design,” in *ACM RecSys '07*, pp. 153–156, ACM, 2007.
- [146] R. Torres, S. M. McNee, M. Abel, J. A. Konstan, and J. Riedl, “Enhancing digital libraries with TechLens+,” in *ACM/IEEE JCDL '04*, pp. 228–236, ACM, 2004.
- [147] M. van Alstyne and E. Brynjolfsson, “Global village or cyber-balkans? Modeling and measuring the integration of electronic communities,” *Management Science*, vol. 51, no. 6, pp. 851–868, June 2005.
- [148] P. Viappiani and C. Boutilier, “Regret-based optimal recommendation sets in conversational recommender systems,” in *ACM RecSys '09*, pp. 101–108, ACM, 2009.

- [149] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*. Princeton, NJ: Princeton University Press, 1944.
- [150] G. Walsh and J. Golbeck, “Curator: A game with a purpose for collection recommendation,” in *ACM CHI ’10*, pp. 2079–2082, ACM, ACM ID: 1753643, 2010.
- [151] P. M. West, D. Ariely, S. Bellman, E. Bradlow, J. Huber, E. Johnson, B. Kahn, J. Little, and D. Schkade, “Agents to the Rescue?,” *Marketing Letters*, vol. 10, no. 3, pp. 285–300, 1999.
- [152] M. Xie, L. V. S. Lakshmanan, and P. T. Wood, “Breaking out of the box of recommendations: From items to packages,” in *ACM RecSys ’10*, pp. 151–158, ACM, ACM ID: 1864739, 2010.
- [153] Y. Yang and X. Liu, “A re-examination of text categorization methods,” in *ACM SIGIR ’99*, pp. 42–49, ACM, 1999.
- [154] C. Ziegler and J. Golbeck, “Investigating interactions of trust and interest similarity,” *Decision Support Systems*, vol. 43, no. 2, pp. 460–475, March 2007.
- [155] C. Ziegler and G. Lausen, “Propagation models for trust and distrust in social networks,” *Information Systems Frontiers*, vol. 7, no. 4, pp. 337–358, December 2005.
- [156] C. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen, “Improving recommendation lists through topic diversification,” in *ACM WWW ’05*, pp. 22–32, ACM, 2005.
- [157] P. Zigoris and Y. Zhang, “Bayesian adaptive user profiling with explicit & implicit feedback,” in *ACM CIKM ’06*, pp. 397–404, ACM, 2006.

# Improved Neighborhood-based Collaborative Filtering

Robert M. Bell and Yehuda Koren  
AT&T Labs – Research  
180 Park Ave, Florham Park, NJ 07932  
[{rbell,yehuda}@research.att.com](mailto:{rbell,yehuda}@research.att.com)

## ABSTRACT

Recommender systems based on collaborative filtering predict user preferences for products or services by learning past user-item relationships. A predominant approach to collaborative filtering is neighborhood based (“ $k$ -nearest neighbors”), where a user-item preference rating is interpolated from ratings of similar items and/or users. In this work, we enhance the neighborhood-based approach leading to a substantial improvement of prediction accuracy, without a meaningful increase in running time. First, we remove certain so-called “global effects” from the data to make the different ratings more comparable, thereby improving interpolation accuracy. Second, we show how to simultaneously derive interpolation weights for all nearest neighbors. Unlike previous approaches where each interpolation weight is computed separately, simultaneous interpolation accounts for the many interactions between neighbors by globally solving a suitable optimization problem, also leading to improved accuracy. Our method is very fast in practice, generating a prediction in about 0.2 milliseconds. Importantly, it does not require training many parameters or a lengthy preprocessing, making it very practical for large scale applications. The method was evaluated on the Netflix dataset. We could process the 2.8 million queries of the Qualifying set in 10 minutes yielding a RMSE of 0.9086. Moreover, when an extensive training is allowed, such as SVD-factorization at the preprocessing stage, our method can produce results with a RMSE of 0.8982.

## Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications—*Data Mining*

## General Terms

Algorithms

## Keywords

collaborative filtering, recommender systems,  $k$  nearest neighbors, matrix factorization

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDDCup'07, August 12, 2007, San Jose, California, USA.  
Copyright 2007 ACM 978-1-59593-834-3/07/0008 ...\$5.00.

## 1. INTRODUCTION

Recommender systems analyze patterns of user interest in items or products to provide personalized recommendations for items that will suit a user’s taste. In essence, recommender systems attempt to profile user preferences and model the interaction between users and products. Increasingly, their excellent ability to characterize and recommend items within huge collections represent a computerized alternative to human recommendations. Because good personalized recommendations can add another dimension to the user experience, some of the largest e-commerce web sites have invested in high-quality recommender systems. Two known examples are the web merchant Amazon.com and the online movie rental company Netflix, which make the recommender system a salient part of their web sites. For a recent survey on recommender systems, refer to [1].

Broadly speaking, recommender systems use either of two strategies. The *content based approach* profiles each user or product allowing programs to associate users with matching products. For example, a movie profile might describe its genre, the participating actors, its box office popularity, etc. User profiles could include demographic information or answers to a suitable questionnaire. Of course, content based strategies require gathering external information that might not be available or easy to collect.

We focus on an alternative strategy that relies only on past user behavior—e.g., their previous transactions or product ratings—and does not require the creation of explicit profiles. This approach is known as *Collaborative Filtering* (CF), a term coined by the developers of the first recommender system - Tapestry [6]. CF analyzes relationships between users and interdependencies among products, in order to identify new user-item associations. Besides avoiding the need for extensive data collection about items or users, CF requires no domain knowledge. In addition, it offers the potential to uncover patterns that would be difficult or impossible to profile using content based techniques. This has led to many papers (e.g., [8]), research projects (e.g., [9]) and commercial systems (e.g., [10]) based on CF.

In a more abstract manner, the CF problem can be cast as missing value estimation: we are given a user-item matrix of scores with many missing values, and our goal is to estimate the missing values based on the given ones. The known user-item scores measure the amount of interest between respective users and items. They can be explicitly given by users that rate their interest in certain items or might be derived from historical purchases. We call these user-item scores *ratings*, and they constitute the input to our algorithm.

The most common form of CF is the neighborhood-based approach (also known as “ $k$  Nearest Neighbors” or kNN, for short). These kNN methods identify pairs of items that tend to be rated similarly or like-minded users with similar history of rating or pur-

chasing, in order to deduce unknown relationships between users and items. Merits of the neighborhood-based approach that have made it popular include: its intuitiveness, sparing the need to train and tune many parameters, and the ability to easily explain to the user the reasoning behind a recommendation.

Three major components characterize the kNN approach: (1) data normalization, (2) neighbor selection, and (3) determination of interpolation weights. Our experience has shown little difference among various neighbor selection strategies (e.g., distance-based vs. correlation-based). However, the two other components, namely data normalization and interpolation weights, have proved vital to the success of the scheme. Accordingly, in this work we revisit these two components and suggest novel methods to accomplish them. We significantly improve the accuracy of kNN approaches without meaningfully affecting running time. Our two main contributions are:

1. It is customary to normalize the data before activating a kNN method. This brings different ratings to a closer level, which allows a better mixing thereof. Usually this is achieved by adjusting for the varying mean ratings across users and/or items. In Section 3 we offer a more comprehensive treatment of this normalization issue, which considers additional effects that are readily available in virtually all ratings datasets. This allows us to explain and eliminate much of the interfering variability from the data and to work with residuals that are more amenable to mutual interpolation.
2. Past kNN methods relate items (or users) by various heuristic variants of correlation coefficients, which allowed direct interpolation from neighbors' scores. In Section 4 we offer a rigorous alternative to these interpolation weights based on global optimization of a cost function pertaining to all weights simultaneously. This results in another improvement of estimation quality with a minor increase in running time.

An encouraging evaluation of the results on the Netflix Prize user-movie dataset [3] is provided in Section 5.

## 2. RELATED WORKS

### 2.1 General Framework

Before discussing previous works, we define our notational conventions. We are given ratings about  $m$  users and  $n$  items, arranged in an  $m \times n$  matrix  $R = \{r_{ui}\}_{1 \leq u \leq m, 1 \leq i \leq n}$ . We anticipate three characteristics of the data that may complicate prediction. First, the numbers of users and items may be very large, as in the Netflix data, with the former likely much larger than the latter. Second, an overwhelming portion of the user-item matrix (e.g., 99%) may be unknown. Third, the pattern of observed data may be very nonrandom, i.e., the amount of observed data may vary by several orders of magnitude among users or among items.

We reserve special indexing letters for distinguishing users from items: for users  $u, v$ , and for items  $i, j, k$ .

### 2.2 Neighborhood-based collaborative filtering

The most common approach to CF is the neighborhood-based approach. Its original form, which was shared by virtually all earlier CF systems, is the user-oriented approach; see [8] for a good analysis. Such user-oriented methods estimate unknown ratings based on recorded ratings of like minded users. More formally, in order to estimate the unknown rating  $r_{ui}$ , we resort to a set of users  $N(u; i)$  that tend to rate similarly to  $u$  ("neighbors"), and that actually rated item  $i$  (i.e.,  $r_{vi}$  is known for each  $v \in N(u; i)$ ).

Then, the estimated value of  $r_{ui}$  is taken as a weighted average of the neighbors' ratings:

$$r_{ui} \leftarrow \frac{\sum_{v \in N(u; i)} s_{uv} r_{vi}}{\sum_{v \in N(u; i)} s_{uv}} \quad (1)$$

The similarities – denoted by  $s_{uv}$  – play a central role here as they are used both for selecting the members of  $N(u; i)$  and for weighting the above average. Common choices are the Pearson correlation coefficient and the closely related cosine similarity. Various methods differ by their how they normalize or center data prior to activating interpolation rule (1). For example, prediction quality can be improved by correcting for user-specific means. We present a more comprehensive treatment to data normalization in Section 3.

An analogous alternative to the user-oriented approach is the item-oriented approach [10, 14]. In those methods, a rating is estimated using known ratings made by the same user on similar items. Now, to estimate the unknown  $r_{ui}$ , we identify a set of neighboring items  $N(i; u)$  that other users tend to rate similarly to their rating of  $i$ . Analogous to above, all items in  $N(i; u)$  must have been rated by  $u$ . Then, in parallel to (1), the estimated value of  $r_{ui}$  is taken as a weighted average of the ratings of neighboring items:

$$r_{ui} \leftarrow \frac{\sum_{j \in N(i; u)} s_{ij} r_{uj}}{\sum_{j \in N(i; u)} s_{ij}} \quad (2)$$

As with the user-user similarities, the item-item similarities (denoted by  $s_{ij}$ ) are typically taken as either correlation coefficients or cosine similarities. Sarwar et al. [14] recommend using an adjusted cosine similarity, where ratings are translated by deducting user-means before computing the cosine similarity. They also found that item-oriented approaches deliver better quality estimates than user-oriented approaches while allowing more efficient computations. This is because, typically, the number of items is significantly lower than the number of users, which allows pre-computing all item-item similarities for retrieval as needed.

Neighborhood-based methods became very popular because they are intuitive and relatively simple to implement. In particular, they do not require tuning many parameters or an extensive training stage. They also provide a concise and intuitive justification for the computed predictions. This enables presenting the user a list of similar items that he or she has previously rated, as the basis for the estimated rating. This way, the user actually understands the reasoning behind the recommendation, allowing him/her to better assess its relevance (e.g., downgrade the estimated rating if it is based on an item that he or she no longer likes), or even encourage the user to alter outdated ratings.

However, neighborhood-based methods raise some concerns:

1. The similarity function ( $s_{uv}$  or  $s_{ij}$ ), which directly defines the interpolation weights, is arbitrary. Different CF algorithms use somewhat different similarity measures; all are trying to quantify the elusive notion of user- or item-similarity. We could not find any fundamental justification for the chosen similarities. Consider an extreme example, where a particular item is predicted perfectly by some other item. In that case, we would want the latter item to receive all the weight, but that is impossible for bounded similarity scores like the Pearson correlation coefficient.
2. Another problem is that previous neighborhood-based methods do not account for interactions among neighbors. Each similarity between an item  $i$  and a neighbor  $j \in N(i; u)$  is computed independently of the content of  $N(i; u)$  and the

other similarities:  $s_{ik}$  for  $k \in N(i; u) - \{j\}$ . For example, suppose that our items are movies, and the neighbors set contains three movies that are highly correlated with each other (e.g., sequels such as “Lord of the Rings 1–3”). An algorithm that ignores the similarity of the three movies when determining their interpolation weights, may end up essentially triple counting the information provided by the group.

3. By definition, the interpolation weights sum to one, which may cause overfitting. Suppose that an item has no useful neighbors rated by a particular user. In that case, it would be best to ignore the neighborhood information, staying with the current data normalization. Nevertheless, the standard neighborhood formula uses a weighted average of ratings for the uninformative neighbors.
4. Neighborhood methods may not work well if variability differs substantially among neighboring items (users)

In a recent work [2], we overcame most of these shortcomings, but the result was a rather slow algorithm, whose running time was several orders of magnitude slower than previous neighborhood-methods, thereby limiting its practical applicability. In this work, we show how to solve the aforementioned issues of neighborhood based approaches, without compromising running time efficiency.

### 2.3 Factorization-based collaborative filtering

Matrix factorization is an alternative approach to CF. It is related to the so-called model-based approaches, which train a compact model that explains the full ratings matrix. It is not directly related to the methods discussed in this paper, but we survey it here because it can be integrated with our methods. In a way, factorization complements the more localized neighborhood-based methods by providing a higher level perspective to the ratings data. Our experiments show that such a combination can improve upon the application of either method alone; see Section 5.

The goal of a factorization-based CF is to uncover latent features of the given data that explain the ratings. This can be achieved by employing matrix factorization techniques such as Singular Value Decomposition (SVD) or Principal Components Analysis (PCA). Given an  $m \times n$  matrix  $R$ , SVD computes the best rank- $f$  approximation  $R^f$ , which is defined as the product of two rank- $f$  matrices  $P_{m \times f}$  and  $Q_{n \times f}$ , where  $f \leq m, n$ . That is,  $R^f = PQ^T$  minimizes the Frobenius norm  $\|R - R^f\|_F$  among all rank- $f$  matrices. In this sense, the matrix  $R^f$  captures the  $f$  most prominent features of the data, leaving out less significant features of the data that might be mere noise. Consequently, each unknown rating,  $r_{ui}$  is estimated as  $R_{ui}^f$ .

Applying an SVD-based technique to CF raises unique difficulties due to the sparsity issue. The conventional SVD computation requires that all entries of  $R$  are known. In fact, the goal of SVD is not properly defined when some entries of  $R$  are missing. Previous works have adapted matrix factorizations techniques to handle missing data in a variety of ways [2, 5, 7, 12, 13]. In all cases, a critical issue is to avoid overfitting for items and users with relatively sparse data.

Factorization-based approaches tend to produce competitive results in terms of accuracy. However, their major drawback is a lengthy training phase that is required for building the model. Whenever these factorization methods are applicable, they can be used in conjunction with the neighborhood-based methods discussed in this paper, as will be explained later in Section 5.

### 3. NORMALIZING BY REMOVING GLOBAL EFFECTS

Although neighborhood and factorization based CF are both powerful prediction methods, there are several reasons to precede either technique with simple models that estimate what we call “global effects.” First, there may be large user and item effects—i.e., systematic tendencies for some users to give higher ratings than other users and for some items to receive higher ratings than others. The basic kNN interpolation method detailed in equations (1)–(2) requires ratings where user and item effects have been taken out in order to, e.g., avoid predicting too high for low-rated items that happen to have a lot of neighbors with high average ratings, and vice versa. Or, to adjust for the fact that some items were mostly rated by users that tend to rate higher, while some other items were rated by users that tend to rate lower.

Second, one may have access to information about either the items or users that can benefit the model. Although factorization offers the potential to detect such structure through estimation of latent variables, directly incorporating variables such as movie genre and user demographics may be more effective and does not require training a factorization model. While such content based analysis is beyond the scope of this paper, we do consider two types of easily derived variables—the number of ratings of an item or by a user and the average rating of an item or by a user. Variables like these allow us, for example, to distinguish users who like the most commonly rated movies best from those who prefer more specialized fare (after controlling for both movie and user effects).

Third, there may be characteristics of specific ratings, such as the date of the rating, that explain some of the variation in scores. For example, a particular user’s ratings may slowly, or suddenly, rise over time, above and beyond any change explained by the inherent quality of the items being rated. Similarly, ratings for some movies may fall with time after their initial release dates, while others stand the test of time quite well. To the extent that patterns like these occur, neither factorization nor kNN could be expected to detect the patterns.

#### 3.1 Methods

Our strategy is to estimate one “effect” at a time, in sequence (i.e., the main effect for items, the main effect for users, a user-time interaction, etc.). At each step, we use residuals from the previous step as the dependent variable for the current step. Consequently, after the first step, the  $r_{ui}$  refer to residuals, rather than raw ratings.

For each of the effects mentioned above, our goal is to estimate either one parameter for each item or one parameter for each user. For the rest of this subsection, we describe our methods for estimating user specific parameters; the method for items is perfectly analogous.

We denote by  $x_{ui}$  the explanatory variable of interest corresponding to user  $u$  and item  $i$ . For user main effects, the  $x_{ui}$ ’s are identically 1. For other global effects, we center  $x_{ui}$  for each user by subtracting the mean of  $x_{ui}$  for that user. In each case, our model is:

$$r_{ui} = \theta_u x_{ui} + \text{error} \quad (3)$$

With sufficient ratings for user  $u$ , we might use the unbiased estimator:

$$\hat{\theta}_u = \frac{\sum_i r_{ui} x_{ui}}{\sum_i x_{ui}^2}$$

where each summation is over all items rated by  $u$ . However, for sparse data, some values of  $\hat{\theta}_u$  may be based on very few observations, thereby resulting in unreliable estimates.

To avoid overfitting, we shrink individual values of  $\hat{\theta}_u$  towards a common value. Shrinkage can be motivated from a Bayesian perspective. Suppose that the true  $\theta_u$  are independent random variables drawn from a normal distribution,

$$\theta_u \sim N(\mu, \tau^2)$$

for known  $\mu$  and  $\tau^2$ , while

$$\hat{\theta}_u | \theta_u \sim N(\theta_u, \sigma_u^2)$$

for known  $\sigma_u^2$ . Then, the best estimator for  $\theta_u$  is its posterior mean:

$$E(\theta_u | \hat{\theta}_u) = \frac{\tau^2 \hat{\theta}_u + \sigma_u^2 \mu}{\tau^2 + \sigma_u^2}$$

a linear combination of the empirical estimator  $\hat{\theta}_u$  and the common mean  $\mu$ . The parameter  $\sigma_u^2$  can be estimated from the formula for the variance of a weighted mean, while  $\mu$  can be estimated by the mean of the  $\theta_u$ 's (optionally weighted by  $n_u$ ). Empirical Bayes [4] suggests that the maximum likelihood estimate of  $\tau^2$  can be found as the solution to:

$$\tau^2 = \frac{\sum_u [(\hat{\theta}_u - \mu)^2 - \sigma_u^2]/(\tau^2 + \sigma_u^2)^2}{\sum_u (\tau^2 + \sigma_u^2)^{-2}}$$

In practice, we used a slightly simpler estimator for  $\theta_u$  by assuming  $\mu = 0$  and  $\sigma_u^2$  is proportional to  $1/n_u$ , which yields:

$$\frac{n_u \hat{\theta}_u}{n_u + \alpha}$$

where  $n_u$  is the number of ratings by user  $u$  and  $\alpha$  is a constant. We determined  $\alpha$  by cross validation.

### 3.2 Example

We illustrate the impact of estimating successive global effects on the Netflix data [3]. The dataset is based on more than 100 million ratings of movies performed by anonymous Netflix customers and is described in details in Section 5. Here, we report results on the *Probe set*, which is a test set containing about 1.4 million ratings compiled by Netflix. Ratings are predicted by accumulating a series of global effects. Accuracy of predictions is measured by their root mean squared error (RMSE). The effects that we describe here should be readily available in every user-item ratings system, and are not particular to the Netflix data.

Table 1 shows how the RMSE for the probe set declines with the inclusion of each successive global effect. Not surprisingly, by far the largest improvements in RMSE are associated with the two sets of main effects—the movie effect and the user effect. These two main effects are comparable to double centering enhanced with shrinkage. They reduce the RMSE from 1.1296 based on use of the mean rating in the training data, down to 0.9841.

Of more interest are various interactions. The first interaction term,  $\text{User} \times \text{Time}(\text{user})^{1/2}$ , refers to allowing each user's rating to change linearly with the square root of the number of days since the user's first rating. Exploratory analysis indicated that this transformation improved the fit relative to the untransformed number of days. This term, which allows for the type of drift hypothesized above, reduces the RMSE by 0.0032. Technically, in (3) we set each  $x_{ui}$  to the square root of number of days elapsed since the first rating by user  $u$  till the time  $u$  rated item  $i$ . Then, for each fixed user  $u$ , we center all computed values of  $x_{ui}$ , and calculate  $\hat{\theta}_u$  in order to regress  $r_{ui}$  on  $x_{ui}$ .

Similarly,  $\text{User} \times \text{Time}(\text{movie})^{1/2}$ , allows each user's rating to change linearly with the square root of the number of days since

| Effect  | RMSE   | Improvement |
|---|--------|-------------|
| Overall mean  | 1.1296 | NA          |
| Movie effect  | 1.0527 | .0769       |
| User effect   | 0.9841 | .0686       |
| $\text{User} \times \text{Time}(\text{user})^{1/2}$   | 0.9809 | .0032       |
| $\text{User} \times \text{Time}(\text{movie})^{1/2}$  | 0.9786 | .0023       |
| $\text{Movie} \times \text{Time}(\text{movie})^{1/2}$ | 0.9767 | .0019       |
| $\text{Movie} \times \text{Time}(\text{user})^{1/2}$  | 0.9759 | .0008       |
| User $\times$ Movie average                           | 0.9719 | .0040       |
| User $\times$ Movie support                           | 0.9690 | .0029       |
| Movie $\times$ User average                           | 0.9670 | .0020       |
| Movie $\times$ User support                           | 0.9657 | .0013       |

**Table 1: RMSE for Netflix probe data after adding a series of global effects to the model**

the movie's first rating by anyone. Somewhat surprisingly, the latter interaction contributes almost as much as the first. Two additional time interactions concentrate on the complementary movie viewpoint. That is, for each fixed movie they model how its ratings change against time. Once again, the time variables are either square root of days since first rating of the movie, or square root of days since first rating by the user.

The next two effects interact users with movie characteristics. We measure the popularity of a movie in two different ways: (1) its average rating; (2) its support, which is the number of ratings associated with the movie. These effects—User  $\times$  Movie average and User  $\times$  Movie support—measure how users change their ratings based on the popularity of the movies. We try to estimate here how closely a user follows the public opinion, or perhaps the opposite, how contrarian is the user. The most effective interaction is between users and movie average—indicating that users varied in terms of how much they agree with the consensus.

Finally, we interact movies with user characteristics, regressing the ratings of each movie on the mean or support of the users. Although these interactions are more difficult to motivate, each contributes an additional modest decrease of the RMSE. Overall, the eight interactions combine to reduce the RMSE by 0.0184 to 0.9657.

## 4. A NEIGHBORHOOD RELATIONSHIPS MODEL

In this section we assume that global effects have already been removed, so whenever we refer to a rating we actually mean the residual remaining after removing global effects from the original rating data. However, our discussion here is completely general, so the following method applies whether global effects are removed or not.

We need to estimate the unknown rating by user  $u$  of item  $i$ , that is  $r_{ui}$ . As with all neighborhood-based methods, our first step is neighbor selection. We focus on an item-oriented method. Among all items rated by  $u$ , we select the  $K$  most similar to  $i$ ,  $N(i; u)$ , by using a similarity function such as the correlation coefficient, as described later in this section. Typical values of  $K$  lie in the range of 20–50; see Section 5.

Given a set of neighbors  $N(i; u)$ , we need to compute *interpolation weights*  $\{w_{ij} | j \in N(i; u)\}$  that will enable the best prediction rule of the form:

$$r_{ui} \leftarrow \sum_{j \in N(i; u)} w_{ij} r_{uj} \quad (4)$$

For notational convenience assume that the  $K$  neighbors in  $N(i; u)$

are indexed by  $1, \dots, K$ , and the corresponding interpolation weights are arranged within  $w \in \mathbb{R}^K$ .

We seek a formal computation of the interpolation weights, that stems directly from their usage within prediction rule (4). As explained earlier, it is important to derive all interpolation weights simultaneously to account for interdependencies among the neighbors. We achieve these goals by defining a suitable optimization problem.

To start, we consider a hypothetical dense case, where all users but  $u$  rated both  $i$  and *all* its neighbors in  $N(i; u)$ . In that case, we could learn the interpolation weights by modeling the relationships between item  $i$  and its neighbors through a least squares problem:

$$\min_w \sum_{v \neq u} \left( r_{vi} - \sum_{j \in N(i; u)} w_{ij} r_{vj} \right)^2 \quad (5)$$

Notice that the only unknowns here are the  $w_{ij}$ 's. The optimal solution to the least squares problem (5) is found by differentiation as a solution of a linear system of equations. From a statistics viewpoint, it is equivalent to the result of a linear regression (without intercept) of  $r_{vi}$  on the  $r_{vj}$  for  $j \in N(i; u)$ . Specifically, the optimal weights are given by:

$$Aw = b \quad (6)$$

Here,  $A$  is a  $K \times K$  matrix defined as:

$$A_{jk} = \sum_{v \neq u} r_{vj} r_{vk} \quad (7)$$

Similarly the vector  $b \in \mathbb{R}^K$  satisfies:

$$b_j = \sum_{v \neq u} r_{vj} r_{vi} \quad (8)$$

For a sparse ratings matrix there are likely to be very few users who rated  $i$  and all its neighbors  $N(i; u)$ . Accordingly, it would be unwise to base  $A$  and  $b$  as given in (7)–(8) only on users with complete data. Even if there are enough users with complete data for  $A$  to be nonsingular, that estimate would ignore a large proportion of the information about pairwise relationships among ratings by the same user. However, we can still estimate  $A$  and  $b$ , up to the same constant, by averaging over the given support, leading to the following reformulation:

$$\bar{A}_{jk} = \frac{\sum_{v \in U(j, k)} r_{vj} r_{vk}}{|U(j, k)|} \quad (9)$$

$$\bar{b}_j = \frac{\sum_{v \in U(i, j)} r_{vj} r_{vi}}{|U(i, j)|} \quad (10)$$

where  $U(j, k)$  is the set of users who rated both items  $j$  and  $k$ .

This is still not enough for overcoming the sparseness issue. The averages represented by  $\bar{A}_{jk}$  or  $\bar{b}_j$  may differ by orders of magnitude in terms of the number of users included in the average. As discussed in the previous section, averages based on relatively low support (small values of  $|U(j, k)|$ ) can generally be improved by shrinkage towards a common mean. Thus, we compute a baseline value which is defined by taking the average of all possible  $\bar{A}_{jk}$  values. Let us denote this baseline value by  $avg$ ; its precise computation is described in the next subsection. Accordingly, we define

the corresponding  $K \times K$  matrix  $\hat{A}$  and the vector  $\hat{b} \in \mathbb{R}^K$ :

$$\hat{A}_{jk} = \frac{|U(j, k)| \cdot \bar{A}_{jk} + \beta \cdot avg}{|U(j, k)| + \beta} \quad (11)$$

$$\hat{b}_j = \frac{|U(i, j)| \cdot \bar{b}_j + \beta \cdot avg}{|U(i, j)| + \beta} \quad (12)$$

The parameter  $\beta$  controls the extent of the shrinkage. A typical value when working with residuals of full global effects is  $\beta = 500$ .

Our best estimate for  $A$  and  $b$  are  $\hat{A}$  and  $\hat{b}$ , respectively. Therefore, we modify (6) so that the interpolation weights are defined as the solution of the linear system:

$$\hat{A}w = \hat{b} \quad (13)$$

The resulting interpolation weights are used within (4) in order to predict  $r_{ui}$ . When working with residuals of global effects, they should be added back to the predicted  $r_{ui}$ , which completes the computation.

Notice that this method addresses all four concerns raised in Section 2.1. First, interpolation weights are derived directly from the ratings, not based on any similarity measure. Second, the interpolation weights formula explicitly accounts for relationships among the neighbors. Third, the sum of the weights is not constrained to equal one. If an item (or user) has only weak neighbors, the estimated weights may all be very small. Fourth, the method automatically adjusts for variations among items in their means or variances.

## 4.1 Preprocessing

An efficient computation of an item-item neighborhood-based method depends on precomputing certain values associated with each movie-movie pair to enable their rapid retrieval.

First, we need a quick access to all item-item similarities, the  $s_{ij}$  values. These similarities are required for identifying the  $K$  neighbors that constitute  $N(i; u)$ . Here, we usually take  $s_{ij}$  as the Pearson correlation coefficient between  $i$  and  $j$  calculated on their common raters. It is important to shrink  $s_{ij}$  based on their support, e.g., multiplying the correlation by:  $|U(i, j)| / (|U(i, j)| + \alpha)$  for some small  $\alpha$ . An alternative, which works similarly well, is based on the mean squared error between items:

$$s_{ij} = \frac{|U(i, j)|}{\sum_{u \in U(i, j)} (r_{ui} - r_{uj})^2 + \alpha}$$

The second set of values that should be precomputed is all possible entries of  $\hat{A}$  and  $\hat{b}$ . To this end, for each two items  $i$  and  $j$ , we compute:

$$\bar{A}_{ij} = \frac{\sum_{v \in U(i, j)} r_{vi} r_{vj}}{|U(i, j)|}$$

Then, the aforementioned baseline value  $avg$ , which is used in (11)–(12), is taken as the average entry of the precomputed  $n \times n$  matrix  $\bar{A}$ . In fact, we recommend using two different baseline values, one by averaging the non-diagonal entries of  $\bar{A}$  and another one by averaging the diagonal entries. This accounts for the fact that the diagonal entries are expected to have an inherently higher average because they sum only non-negative values. Finally, we derive a full  $n \times n$  matrix  $\hat{A}$  from  $\bar{A}$  by (11). Here, the non-diagonal average is used when deriving the non-diagonal entries of  $\hat{A}$ , whereas the diagonal average is used when deriving the diagonal entries of  $\hat{A}$ .

Because of symmetry, it is sufficient to store the values of  $s_{ij}$  and  $\hat{A}_{ij}$  only for  $i \geq j$ . We allocated one byte for each individual

```

NonNegativeQuadraticOpt ( $A \in \mathbb{R}^{K \times K}$ ,  $b \in \mathbb{R}^K$ )
% Minimize  $x^T Ax - 2b^T x$  s.t.  $x \geq 0$ 

do
   $r \leftarrow Ax - b$  % the residual, or "steepest gradient"
  % find active variables - those that are pinned because of
  % nonnegativity constraint, and set respective  $r_i$ 's to zero
  for  $i = 1, \dots, k$  do
    if  $x_i = 0$  and  $r_i < 0$  then
       $r_i \leftarrow 0$ 
    end if
  end for
   $\alpha \leftarrow \frac{r^T r}{r^T A r}$  % max step size
  % adjust step size to prevent negative values:
  for  $i = 1, \dots, k$  do
    if  $r_i < 0$  then
       $\alpha \leftarrow \min(\alpha, -x_i/r_i)$ 
    end if
  end for
   $x \leftarrow x + \alpha r$ 
  while  $\|r\| < \epsilon$  % stop when residual is close to 0
  return  $x$ 

```

**Figure 1: Minimizing a quadratic function with non-negativity constraints**

value, so overall space complexity for  $n$  items is exactly  $n(n + 1)$  bytes. E.g., for the Netflix dataset which contains 17770 movies, overall memory requirements are 300MB, easily fitting within core memory. A more comprehensive system, which includes data on 100,000 items would require about 10GB of space, which can fit within core memory of current 64bit servers. For even larger systems, disk resident storage of item-item values is still practical, as evident by the Amazon item-item recommender system, which is accessing stored similarity information for several million catalog items [10]. Preprocessing time is quadratic in  $n$  and linear in the number of ratings. The time required for computing all  $s_{ij}$ 's and  $\hat{A}_{ij}$ 's on the Netflix data (which contains 100 million ratings) was about 15 minutes on a Pentium 4 PC. Notice that preprocessing can be easily parallelized.

The fact that all possible entries of matrix  $\hat{A}$  are precomputed, saves the otherwise lengthy time needed to construct it. Thus, after quickly retrieving the relevant entries of  $\hat{A}$ , we can compute the interpolation weights by solving a  $K \times K$  system of equations. For typical values of  $K$  (between 20 and 50), this time is comparable to the time needed for computing the  $K$  nearest neighbors, which is common to all neighborhood-based approaches. Hence, while our method relies on a much more detailed computation of the interpolation weights compared to previous methods, it does not significantly increase running time; see Section 5.

## 4.2 Calculation of the weights

The interpolation weights can be computed by solving (13) using standard linear equations solvers. However, we experienced a modest increase in accuracy when  $w$  is constrained to be non-negative, which avoids certain redundant overfitting. This leads to a quadratic program which can be solved by calling “NonNegativeQuadraticOpt( $\hat{A}, \hat{b}$ )”, as described in Figure 1. The function is based on the principles of the Gradient Projection method; see, e.g., [11].

## 4.3 User-oriented implementation

In the above discussion we derived an item-oriented approach, but a parallel idea was successfully applied in a user-oriented fashion, by switching the roles of users and items throughout the above discussion. However, as with previous neighborhood-based approaches, an item-oriented approach enables a much faster implementation by precomputing and storing in main memory a full item-item matrix containing all  $s_{ij}$  and  $\hat{A}_{ij}$  values for future retrieval. Typically, the larger number of users will complicate such a precomputation in terms of both time and space, and out-of-core storage will be required. Moreover, our experience with the Netflix data, under various settings, showed that an item-oriented approach consistently delivered more accurate predictions than a user-oriented one. This advantage of item-oriented methods strengthens a similar observation by Sarwar et al. [14]. However, when a user-oriented approach is computationally feasible, it can be used for improving prediction accuracy by mixing its results with those of the item-oriented approach as in [15].

## 5 EXPERIMENTAL STUDY

We evaluated our algorithms on the Netflix data of more than 100 million movie ratings [3]. We are not aware of any publicly available CF dataset that is close to the scope and quality of this dataset.

To maintain compatibility with results published by others, we adopted some standards that were set by Netflix, as follows. First, quality of the results is measured by their root mean squared error (RMSE), a measure that puts more emphasis on large errors compared with the alternative of mean absolute error. In addition, we report results on two test sets compiled by Netflix, one is known as *the Probe set* and the other is known as *the Quiz set*. These two test sets were constructed similarly, with both containing about 1.4 million of the most recent movie ratings (by date) performed by the users. The true ratings of the Probe set are provided. However, we do not know the true ratings for the Quiz set, which are held by Netflix being the subject of the Netflix Prize contest. Netflix provides RMSE values to competitors that submit their predictions for the Quiz set. The benchmark is Netflix’s proprietary CF system, Cinematch, which achieved a RMSE of 0.9514 on this Quiz set. The two test sets contain many more ratings by users that do not rate much and are harder to predict. In a way, they represent real requirements from a CF system, which needs to predict new ratings from older ones, and to equally address all users, not only the heavy raters.

### 5.1 Experiments with the Probe set

Table 2 compares our method to a kNN method that takes interpolation weights as Pearson correlation coefficients (shrunk to improve results), which represent the common approach to the problem. All methods use item neighborhoods. Reported results are RMSE values computed on the Probe set. We apply the methods after varying stages of preprocessing the data. First, we applied the methods to the raw data, where scores were not normalized at all. Then, we removed the first two effects – the user effect and the item effect, which is similar to double-centering. When no neighborhood-interpolation is performed, such a normalization alone delivers a RMSE of 0.9841, as can be seen in the second column of the table. A more substantial normalization is achieved by accounting for all the global effects, as described in Section 3. Global effects on their own lower the Probe RMSE to 0.9657, before applying any neighborhood interpolation.

Finally, our method can be combined with the factorization approach described in Subsection 2.3. Notice that unlike the previ-

| Data normalization | No interpolation<br>( $k = 0$ ) | Correlation-based interpolation |          |          | Jointly derived interpolation |          |          |
|--------------------|---------------------------------|---------------------------------|----------|----------|-------------------------------|----------|----------|
|                    |                                 | $k = 20$                        | $k = 35$ | $k = 50$ | $k = 20$                      | $k = 35$ | $k = 50$ |
| none (raw scores)  | NA                              | 0.9947                          | 1.002    | 1.0085   | 0.9536                        | 0.9596   | 0.9644   |
| double centering   | 0.9841                          | 0.9431                          | 0.9470   | 0.9502   | 0.9216                        | 0.9198   | 0.9197   |
| global effects     | 0.9657                          | 0.9364                          | 0.9390   | 0.9413   | 0.9194                        | 0.9179   | 0.9174   |
| factorization      | 0.9167                          | 0.9156                          | 0.9142   | 0.9142   | 0.9071                        | 0.9071   | 0.9071   |

**Table 2: Comparing our interpolation scheme against conventional correlation-based interpolation, by reporting RMSEs on the Probe set. Various levels of data normalization (preprocessing) are shown, and different sizes of neighborhoods ( $K$ ) are considered.**

ously mentioned normalizations, factorization requires an extensive training in order to learn the factors. Our implementation delivered a RMSE of 0.9167 on the Probe set, before considering any neighborhood information. A kNN method can be used to improve factorization results, by applying it to the residuals remaining after subtracting the estimates generated by factorization. Effectively, error is smoothed by considering local information that the rather regional factorization approaches tend to miss.

The remaining columns of Table 2 contain results using the two methods for determining interpolation weights. We provide results representing the spectrum of viable choices for neighborhood size –  $K$ . For each value of  $K$ , the same neighborhoods are used for the two methods, the only difference lies in the determination of interpolation weights. The main findings are as follows.

- The jointly derived interpolation weights proposed in this paper uniformly outperformed standard correlation based weights. The improvement was 0.0071 when applied after factorization, but was much larger with less extensive data normalizations.
- Use of the neighborhood model with our proposed interpolation weights substantially improved the predictions across all the data normalizations. The neighborhood approach reduced RMSE by 0.0096 even after the much more time consuming factorization method. Notably, the correlation-based neighborhood model produced a much more modest improvement of 0.0025 relative to factorization.
- The best neighborhood size  $K$  varied depending on the extent of prior data normalization. Interestingly, we found no difference in RMSE over the range 20 to 50 neighbors for the best combination–jointly derived interpolation weights after factorization.
- Even with incorporation of neighborhood information, more complete data normalization improved predictions. However, there were diminishing returns. For example, while global effects reduced RMSE by 0.0184 before the neighborhood model, only 0.0023 survived after incorporation of the neighborhood model.

As for running time, the correlation based method required about 200 seconds to process the whole Probe set, regardless of the value of  $K$ . Our method, with  $K = 20$ , required a slight increase in time to about 235 seconds for the whole Probe set, where the added time represents the effort needed to solve the  $K \times K$  least squares problem for each query. Not surprisingly, increasing  $K$  from 20 to 35 or 50 also raises running time to around 355 seconds ( $K = 35$ ) or 470 seconds ( $K = 50$ ), due to the growing effort needed for solving the larger least squares problems. Even so, the slower mode of our method ( $K = 50$ ) processed a single rating query in less than 0.4 millisecond.

## 5.2 Experiments with the Quiz set

When computing predictions for the Quiz set, the test set held by Netflix, we subsume the Probe set into our training data. This makes Quiz results better than the Probe results. Accordingly, our method produced a RMSE of 0.9086 (4.5% improvement over Netflix system) for the quiz set when applied to residuals of global effects. When applied to residuals of factorization the RMSE dropped to 0.8982 (5.6% improvement over Netflix system), at the cost of requiring the training of a factorization model.

These results are comparable to recently reported results for the Netflix data by methods that require an extensive training phase [2, 12]. Moreover, when mixed with other methods, results can be substantially improved. For example, an alternative method to introducing neighborhood-awareness into factorization-based results was described in [2]. While the accuracy of the two methods is similar, they differ considerably in their nature, and thus produce different ratings. Thus, they can be mixed effectively to produce a more accurate solution. Their mixtures achieve solutions with RMSEs around 0.8900 (6.45% improvement over Netflix system), which would currently rank high on the Netflix Prize Leaderboard<sup>1</sup>, even before mixing with other known approaches.

## 6. SUMMARY

Collaborative filtering through neighborhood-based interpolation (“kNN”) is probably the most popular way to create a recommender system. The success of these methods depends on the choice of the interpolation weights, which are used to estimate unknown ratings from neighboring known ones. Nevertheless, the literature lacks a rigorous way to derive these weights. In this work we showed how the interpolation weights can be computed as a global solution to an optimization problem that precisely reflects their role. Comparison to past kNN methods was made on the Netflix data, with encouraging results suggesting a significant improvement of prediction accuracy without a meaningful increase in running time.

Our second, complementary contribution is related to data normalization. Normalization is essential to kNN methods, as otherwise mixing ratings pertaining to different unnormalized users or items will lead to inferior results. This work offered a comprehensive approach to data normalization, which is related to removing 10 effects that can be readily observed in user-item rating data. Those effects cause much of the data variability and mask the fundamental relationships between ratings. Hence, their removal brings the ratings closer and facilitates improved estimation accuracy.

## 7. REFERENCES

- [1] G. Adomavicius and A. Tuzhilin, “Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions”, *IEEE*

<sup>1</sup>[www.netflixprize.com/leaderboard](http://www.netflixprize.com/leaderboard)

*Transactions on Knowledge and Data Engineering* **17**  
(2005), 634–749.

- [2] R. M. Bell, Y. Koren and C. Volinsky, “Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems”, *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.
- [3] J. Bennet and S. Lanning, “The Netflix Prize”, [www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf](http://www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf).
- [4] B. Efron and C. Morris, “Data analysis using Stein’s estimator and its generalization”, *Journal American Statistical Association* **70** (1975), 311–319.
- [5] S. Funk, “Netflix Update: Try This At Home”, [sifter.org/~simon/journal/20061211.html](http://sifter.org/~simon/journal/20061211.html), 2006.
- [6] D. Goldberg, D. Nichols, B. M. Oki and D. Terry, “Using Collaborative Filtering to Weave an Information Tapestry”, *Communications of the ACM* **35** (1992), 61–70.
- [7] K. Goldberg, T. Roeder, D. Gupta and C. Perkins, “Eigentaste: A Constant Time Collaborative Filtering Algorithm”, *Information Retrieval* **4** (2001), 133–151.
- [8] J. L. Herlocker, J. A. Konstan, A. Borchers and John Riedl, “An Algorithmic Framework for Performing Collaborative Filtering”, *Proc. 22nd ACM SIGIR Conference on Information Retrieval*, pp. 230–237, 1999.
- [9] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon and J. Riedl, “GroupLens: Applying Collaborative Filtering to Usenet News”, *Communications of the ACM* **40** (1997), 77–87, [www.grouplens.org](http://www.grouplens.org).
- [10] G. Linden, B. Smith and J. York, “Amazon.com Recommendations: Item-to-item Collaborative Filtering”, *IEEE Internet Computing* **7** (2003), 76–80.
- [11] J. Nocedal and S. Wright, *Numerical Optimization*, Springer (1999).
- [12] R. Salakhutdinov, A. Mnih, and G. Hinton, “Restricted Boltzmann Machines for Collaborative Filtering”, *Proc. 24th Annual International Conference on Machine Learning*, 2007.
- [13] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, “Application of Dimensionality Reduction in Recommender System – A Case Study”, *WEBKDD’2000*.
- [14] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, “Item-based Collaborative Filtering Recommendation Algorithms”, *Proc. 10th International Conference on the World Wide Web*, pp. 285–295, 2001.
- [15] J. Wang, A. P. de Vries and M. J. T. Reinders, “Unifying User-based and Item-based Collaborative Filtering Approaches by Similarity Fusion”, *Proc. 29th ACM SIGIR Conference on Information Retrieval*, pp. 501–508, 2006.

# Video Lectures

[Help Center](#)

Having trouble viewing lectures? Try changing players. Your current player format is flash. [Change to html5.](#)

## ➤ Module 1 (Week 1)

- ✓ [Video 1.1 - Introduction to Recommender Systems \(38:22\)](#)



- ✓ [Video 1.2 - Welcome to the Course \(24:15\)](#)



- ✓ [Video 1.3a - Software Environment \(6:47\)](#)



- ✓ [Video 1.3b - Setup of Eclipse](#)



- ✓ [Video 1.4a - Taxonomy of Recommender Systems \(27:57\)](#)



- ✓ [Video 1.4b - Taxonomy of Recommender Systems continued \(23:21\)](#)



- ✓ [Video 1.5 - Amazon Tour \(36:04\)](#)



## ➤ Module 2 (Weeks 2 - 3)

- ✓ [Video 2.1 - Introduction to non-personalized recommenders \(32:52\)](#)



- ✓ [Video 2.2 - Preferences and Ratings \(17:21\)](#)



- ✓ [Video 2.3 - Predictions and Recommendations \(16:46\)](#)



- ✓ [Video 2.4 - Scales and Normalization \(20:35\)](#)



- ✓ [Interview with Anthony Jameson - Preferences and Ratings \(14:59\)](#)



## ➤ Module 3 (Weeks 4 -5)

- ✓ [Video 3.1 - Introduction to Content-Based Recommenders \(30:59\)](#)



- ✓ [Video 3.2 - Introduction to LensKit \(36:13\)](#)



- ✓ [Video 3.3 - TFIDF and more! \(24:09\)](#)



- ✓ [Video 3.4 - Content-Based Recommenders in Detail \(29:26\)](#)



✓ Video 3.5 - Tools for Content-Based Filtering (8:53)



✓ Case-Based Reasoning - Interview with Barry Smyth - (13:09)



✓ LensKit Example Walkthrough (27:14)



## > Module 4 (Weeks 6 - 7)

✓ Video 4.1 - Introduction to User-User Collaborative Filtering (20:25)



✓ Video 4.2 - Basic User-User Breakdown (24:15)



✓ Video 4.3 - Variations and Enhancements (33:36)



✓ Video 4.4 - Explaining Recommendations (16:22)



✓ Trust, Reputation, Influence Limiting - Interview with Paul Resnick (21:37)



✓ Trust-Based Recommendation - Interview with Jen Golbeck (15:56)



## > Module 5 (Weeks 8 - 9)

✓ Video 5.1 - Intro to Evaluation (16:47)



✓ Video 5.2 - Basic Accuracy Metrics (20:57)



✓ Video 5.3 - Basic Decision Support Metrics (27:46)



✓ Video 5.4 - Rank Metrics (24:18)



✓ Video 5.5 - Fallacy of Hidden Data Evaluation (15:34)



✓ Video 5.6 - More Metrics (23:39)



✓ Video 5.7 - Experimental Protocols (15:21)



✓ Video 5.8 - Unary Data Evaluation (14:29)



✓ Video 5.9 - Using the LensKit Evaluator (20:39)



✓ Video 5.10 - User-Centered Evaluation (18:18)



## > Module 6 (Weeks 10 - 11)

✓ Video 6.1 - Introduction to Item-Item Collaborative Filtering (20:12)



- ✓ Video 6.2 - Item-Item Algorithm (10:57) 
- ✓ Video 6.3 - Item-Item On Unary Data (10:03) 
- ✓ Video 6.4 - Item-Item Hybrids and Extensions (17:06) 

## > Module 7 (Weeks 12 - 13)

- ✓ Video 7.1 - Introduction to Dimensionality Reduction Recommenders (14:13) 
- ✓ Video 7.2 - Diving Deeper with SVD (19:18) 
- ✓ Video 7.3 - Training SVDs (16:42) 
- ✓ Video 7.4 - FunkSVD Training Algorithm (9:03) 
- ✓ Video 7.5 - Probabilistic Matrix Factorization (17:28) 

## > Module 8 (Week 14)

- ✓ Video 8.1 - Threat Models (12:16) 
- ✓ Video 8.2 - The Cold Start Problem (13:26) 
- ✓ Video 8.3 - What wasn't covered (19:36) 
- ✓ Video 8.4 - More Tools for Recommendation (3:35) 
- ✓ Groups - Interview with Anthony Jameson (14:20) 
- ✓ Context-Aware Recommenders - Interview with Francesco Ricci (20:54) 
- ✓ Learning to Rank - Interview with Xavier Amatriain (20:47) 
- ✓ Video 8.5 - Wrap Up (18:28) 

# Deconstructing Recommender Systems

How Amazon and Netflix predict your preferences and prod you to purchase

By Joseph A. Konstan, John Riedl

Posted 24 Sep 2012 | 14:30 GMT

One morning in April, we each directed our browsers to [Amazon.com](http://amazon.com) (<http://amazon.com>)'s website. Not only did the site greet us by name, the home page opened with a host of suggested purchases. It directed Joe to Barry Greenstein's *Ace on the River: An Advanced Poker Guide*, Jonah Lehrer's *Imagine: How Creativity Works*, and Michael Lewis's *Boomerang: Travels in the New Third World*. For John it selected Dave Barry's *Only Travel Guide You'll Ever Need*, the spy novel *Mission to Paris*, by Alan Furst, and the banking exposé *The Big Short: Inside the Doomsday Machine*, also by Michael Lewis.

By now, online shoppers are accustomed to getting these personalized suggestions. [Netflix](http://Netflix.com) (<http://Netflix.com>) suggests videos to watch. TiVo records programs on its own, just in case we're interested. And [Pandora](http://Pandora.com) (<http://Pandora.com>) builds personalized music streams by predicting what we'll want to listen to.

All of these suggestions come from recommender systems. Driven by computer algorithms, recommenders help consumers by selecting products they will probably like and might buy based on their browsing, searches, purchases, and preferences. Designed to help retailers boost sales, recommenders are a huge and growing business. Meanwhile, the field of recommender system development has grown from a couple of dozen researchers in the mid-1990s to hundreds of researchers today—working for universities, the large online retailers, and dozens of other companies whose sole focus is on these types of systems.

Over the years, recommenders have evolved considerably. They started as relatively crude and often inaccurate predictors of behavior. But the systems improved quickly as more and different types of data about website users became available and they were able to apply innovative algorithms to that data. Today, recommenders are extremely sophisticated and specialized systems that often seem to know you better than you know yourself. And they're expanding beyond retail sites. Universities use them to steer students to courses. Cellphone companies rely on them to predict which users are in danger of switching to another provider. And conference organizers have tested them for assigning papers to peer reviewers.

The two of us have been building and studying recommender systems since their early days, initially as academic researchers working on [the GroupLens Project](http://www.grouplens.org/) (<http://www.grouplens.org/>). Begun in 1992, GroupLens sorted through messages in Usenet discussion forums and pointed users to threads they might be interested in but had not yet discovered on their own. Several years later, we founded [Net Perceptions](http://www.networkworld.com/newsletters/web/0503web2.html) (<http://www.networkworld.com/newsletters/web/0503web2.html>), the leading recommender company during the first Internet boom. Our experience, therefore, gives us a lot of insight into what's going on



Photo-illustration: Christina Beard

behind the scenes at Amazon and other online retailers, even though those companies seldom speak publicly about exactly how their recommendations work. (In this article, our analysis is based on educated observation and deduction, not on any inside information.) Here's what we know.

**Have you ever wondered** what you look like to Amazon? Here is the cold, hard truth: You are a very long row of numbers in a very, very large table. This row describes everything you've looked at, everything you've clicked on, and everything you've purchased on the site; the rest of the table represents the millions of other Amazon shoppers. Your row changes every time you enter the site, and it changes again with every action you take while you're there. That information in turn affects what you see on each page you visit and what e-mail and special offers you receive from the company.

Over the years, the developers of recommender systems have tried a variety of approaches to gather and parse all that data. These days, they've mostly settled on what is called the personalized collaborative recommender. That type of recommender is at the heart of Amazon, Netflix, Facebook's friend suggestions, and [Last.fm](http://www.last.fm/) (<http://www.last.fm/>), a popular music website based in the United Kingdom. They're "personalized" because they track each user's behavior—pages viewed, purchases, and ratings—to come up with recommendations; they aren't bringing up canned sets of suggestions. And they're "collaborative" because they treat two items as being related based on the fact that lots of other customers have purchased or stated a preference for those items, rather than by analyzing sets of product features or keywords.

Personalized collaborative recommenders, in some form or another, have been around since at least 1992. In addition to the GroupLens project, another early recommender was [MIT's Ringo](http://www.sigchi.org/chi95/proceedings/papers/us_bdy.htm) ([http://www.sigchi.org/chi95/proceedings/papers/us\\_bdy.htm](http://www.sigchi.org/chi95/proceedings/papers/us_bdy.htm)), which took lists of albums from users and suggested other music they might like.

GroupLens and Ringo both used a simple collaborative algorithm known as a "user-user" algorithm. This type of algorithm computes the "distance" between pairs of users based on how much they agree on items they have both rated. For instance, if Jim and Jane each give the movie *Tron* five stars, their distance is zero. If Jim then gives *Tron: Legacy* five stars, while Jane rates it three stars, their distance increases. Users whose tastes are relatively "near" each other according to these calculations are said to share a "neighborhood."

But the user-user approach doesn't work that well. For one thing, it's not always easy to form neighborhoods that make sense: Many pairs of users have only a few ratings in common or none at all, and in the case of movies, these few ratings in common tend to be of blockbusters that nearly everyone likes. Also, because the distance between users can change rapidly, user-user algorithms have to do most of their calculations on the spot, and that can take more time than someone clicking around a website is going to hang around.

So most recommenders today rely on an "item-item" algorithm, which calculates the distance between each pair of books or movies or what have you according to how closely users who have rated them agree. People who like books by Tom Clancy are likely to rate books by Clive Cussler highly, so books by Clancy and Cussler are in the same neighborhood. Distances between pairs of items, which may be based on the ratings of thousands or millions of users, tend to be relatively stable over time, so recommenders can precompute distances and generate recommendations more quickly. Both Amazon and Netflix have said publicly that they use variants of an item-item algorithm, though they keep the details secret.

One problem with both user-user and item-item algorithms is the inconsistency of ratings. Users often do not rate the same item the same way if offered the chance to rate it again. Tastes change, moods change, memories fade. MIT conducted one study in the late 1990s that showed an average change of one point on a seven-point scale a year after a user's original rating. Researchers are trying different ways to incorporate such variables into their models; for example, some recommenders will ask users to rerate items when their original ratings seem out of sync with everything else the recommender knows about them.

But the user-user and item-item algorithms have a bigger problem than consistency: They're too rigid. That is, they can spot people who prefer the same item but then miss potential pairs who prefer very similar items. Let's say you're a fan of Monet's water lily paintings. Of the 250 or so paintings of water lilies that the French impressionist did, which is your favorite? Among a group of Monet fans, each person may like a different water lily painting best, but the basic algorithms might not recognize their shared taste for Monet.

About a decade ago, researchers figured out a way to factor in such sets of similar items—a process called dimensionality reduction. This method is much more computationally intensive than the user-user and item-item algorithms, so its adoption has been slower. But as computers have gotten faster and cheaper, it has been gaining ground.

To understand how dimensionality reduction works, let's consider your taste in food and how it compares with that of a million other people. You can represent those tastes in a huge matrix, where each person's taste makes up its own row and each of the thousands of columns is a different food. Your row might show that you gave grilled filet mignon five stars, braised short ribs four and a half stars, fried chicken wings two stars, cold tofu rolls one star, roasted portobello mushroom five stars, steamed edamame with sea salt four stars, and so forth.

A recommender using the matrix wouldn't really care about your particular rating of a particular food, however. Instead, it wants to understand your preferences in general terms, so that it can apply this knowledge to a wide variety of foods. For instance, given the above, the recommender might conclude that you like beef, salty things, and grilled dishes, dislike chicken and anything fried, are neutral on vegetables, and so on. The number of such taste attributes or dimensions would be much smaller than the number of possible foods—there might be 50 or 100 dimensions in all. And by looking at those dimensions, a recommender could quickly determine whether you'd like a new food—say, salt-crusted prime rib—by comparing its dimensions (salty, beef, not chicken, not fried, not vegetable, not grilled) against your profile. This more general representation allows the recommender to spot users who prefer similar yet distinct items. And it substantially compresses the matrix, making the recommender more efficient.

It's a pretty cool solution. But how do you find those taste dimensions? Not by asking a chef. Instead, these systems use a mathematical technique called singular value decomposition to compute the dimensions. The technique involves factoring the original giant matrix into two "taste matrices"—one that includes all the users and the 100 taste dimensions and another that includes all the foods and the 100 taste dimensions—plus a third matrix that, when multiplied by either of the other two, re-creates the original matrix.

Unlike the food example above, the dimensions that get computed are neither describable nor intuitive; they are pure abstract values, and try as you might, you'll never identify one that represents, say, "salty." And that's okay, as long as those values ultimately yield accurate recommendations. The main drawback to this approach is that the time it takes to factor the matrix grows quickly with the number of customers and products—a matrix of 250 million customers and 10 million products would take 1 billion times as long to factor as a matrix of 250 000 customers and 10 000 products. And the process needs to be repeated

frequently. The matrix starts to grow stale as soon as new ratings are received; at a company like Amazon, that happens every second. Fortunately, even a slightly stale matrix works reasonably well. And researchers have been devising new algorithms that provide good approximations to singular value decomposition with substantially faster calculation times.

**By now, you have a basic idea** of how an online retailer sizes you up and tries to match your tastes to those of others whenever you shop at its site. Recommenders have two other features that dramatically affect the recommendations you see: First, beyond figuring out how similar you are to other shoppers, the recommender has to figure out what you actually like. Second, the system operates according to a set of business rules that help ensure its recommendations are both helpful to you and profitable for the retailer.

For example, consider the recommender used for Amazon's online art store, which at last count had more than 9 million prints and posters for sale. [Amazon's art store](http://www.amazon.com/Art-com/b?ie=UTF8&node=3489231) (<http://www.amazon.com/Art-com/b?ie=UTF8&node=3489231>) assesses your preferences in a few ways. It asks you to rate particular artworks on a five-star scale, and it also notes which paintings you enlarge, which you look at multiple times, which you place on a wish list, and which you actually buy. It also tracks which paintings are on your screen at the time as well as others you look at during your session. The retailer uses the path you've traveled through its website—the pages you've viewed and items you've clicked on—to suggest complementary works, and it combines your purchase data with your ratings to build a profile of your long-term preferences.

Companies like Amazon collect an immense amount of data like this about their customers. Nearly any action taken while you are logged in is stored for future use. Thanks to browser cookies, companies can even maintain records on anonymous shoppers, eventually linking the data to a customer profile when the anonymous shopper creates an account or signs in. This explosion of data collection is not unique to online vendors—Walmart is famous for its extensive mining of cash register receipt data. But an online shop is much better positioned to view and record not just your purchases but what items you considered, looked at, and rejected. Throughout much of the world, all of this activity is fair game; only in Europe do data privacy laws restrict such practices to a degree.

Of course, regardless of the law, any customer will react badly if his or her data is used inappropriately. Amazon learned this lesson the hard way back in September 2000, when certain customers discovered they were being quoted higher prices because the website had identified them as regular customers, rather than as

The screenshot shows the product page for '2001: A Space Odyssey'. At the top, there is a large image of the movie poster. Below it, the product details include: Rating: 4.5 stars (1,126 reviews), Starring: Keir Dullea, Gary Lockwood, Directed by: Stanley Kubrick, Runtime: 2 hours 29 minutes, Release year: 1968, Studio: Warner Bros., and a 'Play trailer' button. The main content area is divided into sections: 'Customers Who Viewed This Item Also Viewed' (with items like '2010', 'Dr Strangelove Or: How I Learned To Stop Worrying ...', and 'A Clockwork Orange'), 'Customers Also Bought Items By' (with authors like Stanley Kubrick, Catherine Wheatley, and Scott Bakemman), and 'What Other Items Do Customers Buy After Viewing This Item?' (with items like 'The Stanley Kubrick Archives', '2001: Filming the Future', and '2001: A Space Odyssey'). There is also a section for 'Tags Customers Associate with This Product' where users can add tags like 'space adventure', 'highly recommended', 'best motion picture', etc. A note says 'You tagged this product with "space adventure". Click again to Undo'.

(/img/recommendf2-  
1348086297597.jpg)

**Recommendation odyssey:**  
An Amazon user interested in  
2001: A Space Odyssey sees  
suggestions from three different  
collaborative recommenders.  
*Click on the image for the full  
illustration view.*

shoppers who had entered anonymously or from a comparison-shopping site. Amazon claimed this was just a random price test and the observed relationship to being a regular customer was coincidental, but it nevertheless stopped the practice.

The business rules around these systems are designed to prevent recommenders from making foolish suggestions and also to help online retailers maximize sales without losing your trust. At their most basic level, these systems avoid what's known as the supermarket paradox. For example, nearly everyone who walks into a supermarket likes bananas and will often buy some. So shouldn't the recommender simply recommend bananas to every customer? The answer is no, because it wouldn't help the customer, and it wouldn't increase banana sales. So a smart supermarket recommender will always include a rule to explicitly exclude recommending bananas.

That example may sound simplistic, but in one of our early experiences, our system kept recommending the Beatles' "White Album" to nearly every visitor. Statistically this was a great recommendation: The customers had never purchased this item from the e-commerce site, and most customers rated it highly. And yet the recommendation was useless. Everyone who was interested in the "White Album" already owned a copy.

Most recommender rules are more subtle, of course. When John recently searched for an action movie on Netflix, for instance, he wasn't offered *The Avengers*, because the blockbuster was not yet available for rental, and so the suggestion wouldn't have profited Netflix. Instead it steered him to *Iron Man 2*, which was available for streaming.

Other business rules prevent recommenders from suggesting loss leaders—products that sell below cost to draw people into the site—or conversely encourage them to recommend products that are overstocked. During our time at Net Perceptions, we worked with a client who did just that: He used his recommender system to identify—with considerable success—potential customers for his overstocked goods.

This kind of thing quickly gets tricky, however. A system that simply pushes high-margin products isn't going to earn the customers' trust. It's like going to a restaurant where the waiter steers you toward a particular fish dish. Is it really his favorite? Or did the chef urge the staff to push out the fish before its sell-by date?

To build trust, the more sophisticated recommender systems strive for some degree of transparency by giving customers an idea of why a particular item was recommended and letting them correct their profiles if they don't like the recommendations they're getting.

You can, for instance, delete information from your Amazon profile about things you purchased as gifts; after all, those don't reflect your tastes. You can also find out why certain products have been offered through the recommender. After Amazon selected Jonathan Franzen's novel *Freedom* for John, he clicked on the link labeled "Explain." He then got a brief explanation that certain books on John's Amazon wish list had triggered the recommendation. But as John hadn't read any of the wish list books, he discounted the *Freedom* suggestion. Explanations like these let users know how reliable a given recommendation is.

But profile adjustments and explanations often aren't enough to keep a system on track. Recently Amazon bombarded Joe with e-mails for large-screen HDTVs—as many as three a week for months. Besides sending him more e-mail on the topic than he could possibly want, the retailer didn't recognize that he'd already purchased a TV through his wife's account. What's more, the e-mails did not offer an obvious way for Joe to say, "Thanks, but I'm not interested." Eventually, Joe unsubscribed from certain Amazon e-mails; he doesn't miss the messages, and he has more time to actually watch that TV.

**So how well do recommenders** ultimately work? They certainly are increasing online sales; analyst Jack Aaronson of the Aaronson Group estimates that investments in recommenders bring in returns of 10 to 30 percent, thanks to the increased sales they drive. And they still have a long way to go.

Right now the biggest challenge for those of us who study recommender systems is to figure out how best to judge the new approaches and algorithms. It's not as simple as benchmarking a microprocessor, because different recommenders have very different goals.

The easiest way to evaluate an algorithm is to look at the difference between its predictions and the actual ratings users give. For instance, if John gives the teen-romance novel *Twilight* one star, Amazon might note that it had predicted he would give it two stars, based on the ratings of other similar users, and so its recommender was off by a star. But sellers care much more about errors on highly rated items than errors on low-rated items, because the highly rated items are the ones users are more likely to buy; John is never going to purchase *Twilight*, so scoring this rating contributes little to understanding how well the recommender works.

Another common measure is the extent to which recommendations match actual purchases. This analysis can also be misleading, however, because it erroneously rewards the recommender for items users managed to find on their own—precisely the items they don't need recommendations for!

Given the shortcomings of these approaches, researchers have been working on new metrics that look not just at accuracy but also at other attributes, such as serendipity and diversity.

Serendipity rewards unusual recommendations, particularly those that are valuable to one user but not as valuable to other similar users. An algorithm tuned to serendipity would note that the "White Album" appears to be a good recommendation for nearly everyone and would therefore look for a recommendation that's less common—perhaps Joan Armatrading's *Love and Affection*. This less-popular recommendation wouldn't be as likely to hit its target, but when it did, it would be a much happier surprise to the user.

Looking at the diversity of a recommender's suggestions is also revealing. For instance, a user who loves Dick Francis mysteries might nevertheless be disappointed to get a list of recommendations all written by Dick Francis. A truly diverse list of recommendations could include books by different authors and in different genres, as well as movies, games, and other products.

**Recommender systems research** has all sorts of new ground to break, far beyond fine-tuning existing systems. Researchers today are considering to what extent a recommender should help users explore parts of a site's collection they haven't looked into—say, sending book buyers over to Amazon's clothing department rather than recommending safe items they may be more comfortable with. Going beyond the retail world, recommenders could help expose people to new ideas; even if we disagree with some of them, the overall effect might be positive in that it would help reduce the balkanization of society. Whether recommenders can do that without annoying us or making us distrustful remains to be seen.

But one thing is clear: Recommender systems are only going to get better, collect more data about you, and show up in new and surprising places. And as for you, if you liked this article, Amazon will be happy to recommend entire books on recommender systems that you might also like.

*This article originally appeared in print as "Recommended for You."*

## About the Authors

Joseph A. Konstan and John Riedl (<http://www.grouplens.org/team>) ARE both professors of computer science at the University of Minnesota. As codirectors of GroupLens Research, Konstan, an IEEE Senior Member, and Riedl, an IEEE Fellow, helped create the MovieLens (<http://movielens.umn.edu/login>) recommender system. The pair's scariest recommender moment came during an interview on "ABC Nightline." Just before a station break, MovieLens pitched the 1950s film noir Sunset Boulevard to host Robert Krulwich. Konstan and Riedl had to wait until they were back on the air to hear his verdict on the suggestion: He loved it!

# Factor in the Neighbors: Scalable and Accurate Collaborative Filtering

Yehuda Koren  
AT&T Labs – Research  
180 Park Ave, Florham Park, NJ 07932  
yehuda@research.att.com

## ABSTRACT

Recommender systems provide users with personalized suggestions for products or services. These systems often rely on Collaborating Filtering (CF), where past transactions are analyzed in order to establish connections between users and products. The most common approach to CF is based on neighborhood models, which is based on similarities between products or users. In this work we introduce a new neighborhood model with an improved prediction accuracy. The model works by minimizing a global cost function. Further accuracy improvements are achieved by extending the model to exploit both explicit and implicit feedback by the users. Past models were limited by the need to compute all pairwise similarities between items or users, which grow quadratically with input size. In particular, this limitation vastly complicates adopting user similarity models, due to the typical large number of users. Our new model solves these limitations by factoring the neighborhood model, thus making both item-item and user-user implementations scale linearly with the size of the data. The methods are tested on the Netflix data, with encouraging results. In addition, we suggest a new evaluation metric, which highlights the differences among methods, based on their performance at a top-K recommendation task. Our study reveals a very significant improvement in quality of top-K recommendation.

## 1. INTRODUCTION

<sup>1</sup> Modern consumers are inundated with choices. Electronic retailers and content providers offer a huge selection of products, with unprecedented opportunities to meet a variety of special needs and tastes. Matching consumers with most appropriate products is not trivial, yet it is a key in enhancing user satisfaction and loyalty. This emphasizes the prominence of *recommender systems*, which provide personalized recommendations for products that suit a user’s taste [1]. Internet leaders like Amazon [18], Google [9], Netflix [6], TiVo [2] and Yahoo! [21] are increasingly adopting such recommenders.

Recommender systems are often based on *Collaborative Filtering* (CF) [12], which relies only on past user behavior—e.g., their previous transactions or product ratings—and does not require the creation of explicit profiles. Notably, CF techniques require no domain knowledge and avoid the need for extensive data collection. In addition, relying directly on user behavior allows uncovering complex and unexpected patterns that would be difficult or impossible to profile using known data attributes. As a consequence, CF attracted much of attention in the past decade, resulting in significant progress and being adopted by some successful commercial systems, including Amazon [18], TiVo [2] and Netflix.

In order to establish recommendations, CF systems need to com-

pare fundamentally different objects: items against users. There are two primary approaches to facilitate such a comparison, which constitute the two main disciplines of CF: *the neighborhood approach* and *latent factor models*.

Neighborhood methods are centered on computing the relationships between items or, alternatively, between users. An item-item approach evaluates the preference of a user to an item based on ratings of similar items by the same user. In a sense, these methods transform users to the item space by viewing them as baskets of rated items. This way, we no longer need to compare users to items, but rather directly relate items to items.

Latent factor models, such as Singular Value Decomposition (SVD), comprise an alternative approach by transforming both items and users to the same latent factor space, thus making them directly comparable. The latent space tries to explain ratings by characterizing both products and users on factors automatically inferred from user feedback. For example, when the products are movies, factors might measure obvious dimensions such as comedy vs. drama, amount of action, or orientation to children; less well defined dimensions such as depth of character development or “quirkiness”; or completely uninterpretable dimensions. For users, each factor measures how much the user likes movies that score high on the corresponding movie factor.

Latent factor models offer high expressive ability to describe various aspects of the data. Thus, they tend to provide more accurate results than neighborhood models; see, e.g., [4, 7, 15, 17, 22, 27]. However, most literature and commercial systems (e.g., those of Amazon [18], TiVo [2] and Netflix<sup>2</sup>) are based on the neighborhood models. The prevalence of neighborhood models is partly thanks to their relative simplicity and intuitiveness. However, there are more important reasons for real life systems to stick with those less accurate models. First, they naturally provide intuitive explanations of the reasoning behind recommendations, which often enhance user experience beyond what improved accuracy may achieve. Second, they can immediately provide recommendations based on just entered user feedback.

In this work we introduce a new neighborhood model with an improved accuracy on par with recent latent factor models. In the same time, the model retains the two aforementioned fundamental advantages of neighborhood models. Namely, it can explain its recommendations and handle new ratings without requiring re-training. Admittedly, the new model is not as simple as common neighborhood models, because it requires a training stage much like latent factor models. This reflects a more principled way to modeling neighborhood relations, which rewards the model not only with improved accuracy, but also with greater flexibility to address different aspects of the data and to integrate other models. In particular, we can make the model significantly more scalable than

<sup>1</sup>Parts of this work overlap a KDD’08 publication [17].

<sup>2</sup>Based on personal knowledge.

previous neighborhood models, without compromising its accuracy or other desired properties.

The CF field has enjoyed a surge of interest since October 2006, when the Netflix Prize competition [6] commenced. Netflix released a dataset containing 100 million movie ratings and challenged the research community to develop algorithms that could beat the accuracy of its recommendation system, Cinematch.

A lesson that we learnt through this competition is the importance of integrating different forms of user input into the models [4]. Recommender systems rely on different types of input. Most convenient is the high quality *explicit feedback*, which includes explicit input by users regarding their interest in products. For example, Netflix collects star ratings for movies and TiVo users indicate their preferences for TV shows by hitting thumbs-up/down buttons. However, explicit feedback is not always available. Thus, recommenders can infer user preferences from the more abundant *implicit feedback*, which indirectly reflect opinion through observing user behavior [20]. Types of implicit feedback include purchase history, browsing history, search patterns, or even mouse movements. For example, a user that purchased many books by the same author probably likes that author. Our main focus is on cases where explicit feedback is available. Nonetheless, we recognize the importance of implicit feedback, which can illuminate users that did not provide enough explicit feedback. Hence, our models integrate explicit and implicit feedback.

The structure of the rest of the paper is as follows. We start with preliminaries and related work in Sec. 2. Then, we describe a new, more accurate neighborhood model in Sec. 3. The new model is based on an optimization framework that allows smooth integration with other models, and also inclusion of implicit user feedback. Section 4 shows how to make the model highly scalable by introducing novel factorization techniques. Relevant experimental results are brought within each section. In addition, we suggest a new methodology to evaluate effectiveness of the models, as described in Sec. 5, with encouraging results.

## 2. PRELIMINARIES

We are given ratings about  $m$  users and  $n$  items. We reserve special indexing letters for distinguishing users from items: for users  $u, v$ , and for items  $i, j$ . A rating  $r_{ui}$  indicates the preference by user  $u$  of item  $i$ , where high values mean stronger preference. For example, values can be integers ranging from 1 (star) indicating no interest to 5 (stars) indicating a strong interest. We distinguish predicted ratings from known ones, by using the notation  $\hat{r}_{ui}$  for the predicted value of  $r_{ui}$ . Usually the vast majority of ratings are unknown. For example, in the Netflix data 99% of the possible ratings are missing because a user typically rates only a small portion of the movies. The  $(u, i)$  pairs for which  $r_{ui}$  is known are stored in the set  $\mathcal{K} = \{(u, i) \mid r_{ui} \text{ is known}\}$ . In order to combat overfitting the sparse rating data, models are regularized so estimates are shrunk towards baseline defaults. Regularization is controlled by constants which are denoted as:  $\lambda_1, \lambda_2, \dots$ . Exact values of these constants are determined by cross validation. As they grow, regularization becomes heavier.

### 2.1 Baseline estimates

Typical CF data exhibit large user and item effects – i.e., systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others. It is customary to adjust the data by accounting for these effects, which we encapsulate within the *baseline estimates*. Denote by  $\mu$  the overall average rating. A baseline estimate for an unknown rating  $r_{ui}$  is

denoted by  $b_{ui}$  and accounts for the user and item effects:

$$b_{ui} = \mu + b_u + b_i \quad (1)$$

The parameters  $b_u$  and  $b_i$  indicate the observed deviations of user  $u$  and item  $i$ , respectively, from the average. For example, suppose that we want a baseline estimate for the rating of the movie *Titanic* by user Joe. Now, say that the average rating over all movies,  $\mu$ , is 3.7 stars. Furthermore, *Titanic* is better than an average movie, so it tends to be rated 0.5 stars above the average. On the other hand, Joe is a critical user, who tends to rate 0.3 stars lower than the average. Thus, the baseline estimate for *Titanic*'s rating by Joe would be 3.9 stars by calculating  $3.7 - 0.3 + 0.5$ . In order to estimate  $b_u$  and  $b_i$  one can solve the least squares problem:

$$\min_{b_*} \sum_{(u,i) \in \mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 (\sum_u b_u^2 + \sum_i b_i^2)$$

Here, the first term  $\sum_{(u,i) \in \mathcal{K}} (r_{ui} - \mu + b_u + b_i)^2$  strives to find  $b_u$ 's and  $b_i$ 's that fit the given ratings. The regularizing term  $-\lambda_1 (\sum_u b_u^2 + \sum_i b_i^2)$  – avoids overfitting by penalizing the magnitudes of the parameters.

An easier, yet somewhat less accurate way to estimate the parameters is by decoupling the calculation of the  $b_i$ 's from the calculation of the  $b_u$ 's. First, for each item  $i$  we set:

$$b_i = \frac{\sum_{u:(u,i) \in \mathcal{K}} (r_{ui} - \mu)}{\lambda_2 + |\{u|(u, i) \in \mathcal{K}\}|}$$

Then, for each user  $u$  we set:

$$b_u = \frac{\sum_{i:(u,i) \in \mathcal{K}} (r_{ui} - \mu - b_i)}{\lambda_3 + |\{i|(u, i) \in \mathcal{K}\}|}$$

Averages are shrunk towards zero by using the regularization parameters,  $\lambda_2, \lambda_3$ , which are determined by cross validation. Typical values on the Netflix dataset are:  $\lambda_2 = 25, \lambda_3 = 10$ .

### 2.2 Neighborhood models

The most common approach to CF is based on neighborhood models. Its original form, which was shared by virtually all earlier CF systems, is user-user based; see [14] for a good analysis. Such user-user methods estimate unknown ratings based on recorded ratings of like-minded users. Later, an analogous item-item approach [18, 26] became popular. In those methods, a rating is estimated using known ratings made by the same user on similar items. Better scalability and improved accuracy make the item-item approach more favorable in many cases [3, 26, 27]. In addition, item-item methods are more amenable to explaining the reasoning behind predictions. This is because users are familiar with items previously preferred by them, but do not know those allegedly like-minded users. We focus mostly on item-item approaches, but techniques developed in this work make the user-user approach as computationally efficient; sec Sec. 4.1.

Central to most item-item approaches is a similarity measure between items. Frequently, it is based on the Pearson correlation coefficient,  $\rho_{ij}$ , which measures the tendency of users to rate items  $i$  and  $j$  similarly. Since many ratings are unknown, it is expected that some items share only a handful of common raters. Computation of the correlation coefficient is based only on the common user support. Accordingly, similarities based on a greater user support are more reliable. An appropriate similarity measure, denoted by  $s_{ij}$ , would be a shrunk correlation coefficient:

$$s_{ij} \stackrel{\text{def}}{=} \frac{n_{ij}}{n_{ij} + \lambda_4} \rho_{ij} \quad (2)$$

The variable  $n_{ij}$  denotes the number of users that rated both  $i$  and  $j$ . A typical value for  $\lambda_4$  is 100. Notice that the literature suggests additional alternatives for a similarity measure [26, 27].

Our goal is to predict  $r_{ui}$  – the unobserved rating by user  $u$  for item  $i$ . Using the similarity measure, we identify the  $k$  items rated by  $u$ , which are most similar to  $i$ . This set of  $k$  neighbors is denoted by  $S^k(i; u)$ . The predicted value of  $r_{ui}$  is taken as a weighted average of the ratings of neighboring items, while adjusting for user and item effects through the baseline estimates:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in S^k(i; u)} s_{ij} (r_{uj} - b_{uj})}{\sum_{j \in S^k(i; u)} s_{ij}} \quad (3)$$

Neighborhood-based methods of this form became very popular because they are intuitive and relatively simple to implement. They also offer the following two useful properties:

1. *Explainability.* Users expect a system to give a reason for its predictions, rather than facing “black box” recommendations. This not only enriches the user experience, but also encourages users to interact with the system, fix wrong impressions and improve long-term accuracy. In fact, the importance of explaining automated recommendations is widely recognized [13, 28]. The neighborhood framework allows identifying which of the past user actions are most influential on the computed prediction. In prediction rule (3), each past rating ( $r_{uj}$  for  $j \in S^k(i; u)$ ) receives a separate term in forming the predicted  $\hat{r}_{ui}$ , and thus we can isolate its unique contribution. The past ratings associated with highest contribution are identified as the major explanation behind the recommendation.
2. *New ratings.* Item-item neighborhood models can provide updated recommendations immediately after users entered new ratings. This includes handling new users as soon as they provide feedback to the system, without needing to retrain the model and estimate new parameters. Here, we assume that relationships between items (the  $s_{ij}$  values) are stable and barely change on a daily basis. Thus, prediction rule (3) can be reliably used with the most recent set of ratings given by the user. Notice that for items new to the system we do have to learn new parameters. Interestingly, this asymmetry between users and items meshes well with common practices: systems need to provide immediate recommendations to new users (or new ratings by old users) who expect quality service. On the other hand, it is reasonable to require a waiting period before recommending items new to the system.

However, in a recent work [3], we raised a few concerns about traditional neighborhood schemes. Most notably, these methods are not justified by a formal model. We also questioned the suitability of a similarity measure that isolates the relations between two items, without analyzing the interactions within the full set of neighbors. In addition, the fact that interpolation weights in (3) sum to one forces the method to fully rely on the neighbors even in cases where neighborhood information is absent (i.e., user  $u$  did not rate items similar to  $i$ ), and it would be preferable to rely on baseline estimates.

This led us to propose a more accurate neighborhood model, which overcomes these difficulties. Given a set of neighbors  $S^k(i; u)$  we need to compute *interpolation weights*  $\{\theta_{ij}^u | j \in S^k(i; u)\}$  that enable the best prediction rule of the form:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in S^k(i; u)} \theta_{ij}^u (r_{uj} - b_{uj}) \quad (4)$$

Derivation of the interpolation weights can be done efficiently by estimating all inner products between item ratings; for a full description refer to [3].

### 2.3 Latent factor models

Latent factor models comprise an alternative approach to Collaborative Filtering with the more holistic goal to uncover latent features that explain observed ratings; examples include pLSA [15], neural networks [23], Latent Dirichlet Allocation [8], or models that are induced by Singular Value Decomposition (SVD) on the user-item ratings matrix. Recently, SVD models have gained popularity, thanks to their attractive accuracy and scalability. A typical model associates each user  $u$  with a user-factors vector  $p_u \in \mathbb{R}^f$ , and each item  $i$  with an item-factors vector  $q_i \in \mathbb{R}^f$ . The prediction is done by taking an inner product, i.e.,  $\hat{r}_{ui} = b_{ui} + p_u^T q_i$ . The more involved part is parameter estimation.

In information retrieval it is well established to harness SVD for identifying latent semantic factors [10]. However, applying SVD in the CF domain raises difficulties due to the high portion of missing ratings. Conventional SVD is undefined when knowledge about the matrix is incomplete. Moreover, carelessly addressing only the relatively few known entries is highly prone to overfitting. Earlier works [16, 25] relied on imputation to fill in missing ratings and make the rating matrix dense. However, imputation can be very expensive as it significantly increases the amount of data. In addition, the data may be considerably distorted due to inaccurate imputation. Hence, more recent works [5, 7, 11, 17, 22, 23, 27] suggested modeling directly only the observed ratings, while avoiding overfitting through an adequate regularized model. The reported results compare very favorably with neighborhood models.

However, Latent factor models such as SVD face real difficulties when needed to explain predictions. After all, a key to these models is abstracting users via an intermediate layer of user factors. This intermediate layer separates the computed predictions from past user actions and complicates explanations. Similarly, reflecting new ratings requires re-learning the user factors, and cannot be done at virtually no cost as in the neighborhood models. Thus, we believe that for practical applications neighborhood models are still expected to be a common choice.

### 2.4 The Netflix data

We evaluated our algorithms on the Netflix data of more than 100 million movie ratings performed by anonymous Netflix customers [6]. We are not aware of any publicly available CF dataset that is close to the scope and quality of this dataset. To maintain compatibility with results published by others, we adopted some standards that were set by Netflix, as follows. First, quality of the results is usually measured by their root mean squared error (RMSE):

$$\sqrt{\sum_{(u,i) \in TestSet} (r_{ui} - \hat{r}_{ui})^2 / |TestSet|}$$

a measure that puts more emphasis on large errors compared with the alternative of mean absolute error. In addition, we report results on a test set provided by Netflix (also known as the Quiz set), which contains over 1.4 million recent ratings. Netflix compiled another 1.4 million recent ratings into a validation set, known as the Probe set, which we employ in Section 5. The two sets contain many more ratings by users that do not rate much and are harder to predict. In a way, they represent real requirements from a CF system, which needs to predict new ratings from older ones, and to equally address all users, not only the heavy raters.

The Netflix data is part of the ongoing Netflix Prize competition, where the benchmark is Netflix’s proprietary system, Cinematch,

which achieved a RMSE of 0.9514 on the test set. The grand prize will be awarded to a team that manages to drive this RMSE below 0.8563 (10% improvement).

## 2.5 Implicit feedback

As stated earlier, an important goal of this work is devising models that allow integration of explicit and implicit user feedback. For a dataset such as the Netflix data, the most natural choice for implicit feedback would probably be movie rental history, which tells us about user preferences without requiring them to explicitly provide their ratings. However, such data is not available to us. Nonetheless, a less obvious kind of implicit data does exist within the Netflix dataset. The dataset does not only tell us the rating values, but also *which* movies users rate, regardless of *how* they rated these movies. In other words, a user implicitly tells us about her preferences by choosing to voice her opinion and vote a (high or low) rating. This reduces the ratings matrix into a binary matrix, where “1” stands for “rated”, and “0” for “not rated”. Admittedly, this binary data is not as vast and independent as other sources of implicit feedback could be. Nonetheless, we have found that incorporating this kind of implicit data – which inherently exist in every rating based recommender system – significantly improves prediction accuracy. Some prior techniques, such as Conditional RBMs [23], also capitalized on the same binary view of the data. The benefit of using the binary data is closely related to the fact that ratings do not miss at random, but users are deliberate in which items they choose to rate; see Marlin et al. [19].

The models that we suggest are not limited to a certain kind of implicit data. To keep generality, each user  $u$  is associated with two sets of items, one is denoted by  $R(u)$ , and contains all the items for which ratings by  $u$  are available. The other one, denoted by  $N(u)$ , contains all items for which  $u$  provided an implicit preference.

## 3. A NEIGHBORHOOD MODEL

In this section we introduce a new neighborhood model, which allows an efficient global optimization scheme. The model offers improved accuracy and is able to integrate implicit user feedback. We will gradually construct the various components of the model, through an ongoing refinement of our formulations.

Previous models were centered around *user-specific* interpolation weights –  $\theta_{ij}^u$  in (4) or  $s_{ij}/\sum_{j \in S^k(i;u)} s_{ij}$  in (3) – relating item  $i$  to the items in a user-specific neighborhood  $S^k(i;u)$ . In order to facilitate global optimization, we would like to abandon such user-specific weights in favor of global weights independent of a specific user. The weight from  $j$  to  $i$  is denoted by  $w_{ij}$  and will be learnt from the data through optimization. An initial sketch of the model describes each rating  $r_{ui}$  by the equation:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in R(u)} (r_{uj} - b_{uj})w_{ij} \quad (5)$$

For now, (5) does not look very different from (4), besides our claim that the  $w_{ij}$ 's are not user specific. Another difference, which will be discussed shortly, is that here we sum over all items rated by  $u$ , unlike (4) that sums over members of  $S^k(i;u)$ .

Let us consider the interpretation of the weights. Usually the weights in a neighborhood model represent interpolation coefficients relating unknown ratings to existing ones. Here, it is useful to adopt a different viewpoint, where weights represent offsets to baseline estimates. Now, the residuals,  $r_{uj} - b_{uj}$ , are viewed as the coefficients multiplying those offsets. For two related items  $i$  and  $j$ , we expect  $w_{ij}$  to be high. Thus, whenever a user  $u$  rated  $j$  higher than expected ( $r_{uj} - b_{uj}$  is high), we would like to increase our es-

timate for  $u$ 's rating of  $i$  by adding  $(r_{uj} - b_{uj})w_{ij}$  to the baseline estimate. Likewise, our estimate will not deviate much from the baseline by an item  $j$  that  $u$  rated just as expected ( $r_{uj} - b_{uj}$  is around zero), or by an item  $j$  that is not known to be predictive on  $i$  ( $w_{ij}$  is close to zero). This viewpoint suggests several enhancements to (5). First, we can use implicit feedback, which provide an alternative way to learn user preferences. To this end, we add another set of weights, and rewrite (5) as:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in R(u)} (r_{uj} - b_{uj})w_{ij} + \sum_{j \in N(u)} c_{ij} \quad (6)$$

Much like the  $w_{ij}$ 's, the  $c_{ij}$ 's are offsets added to baseline estimates. For two items  $i$  and  $j$ , an implicit preference by  $u$  to  $j$  lead us to modify our estimate of  $r_{ui}$  by  $c_{ij}$ , which is expected to be high if  $j$  is predictive on  $i$ .<sup>3</sup>

Viewing the weights as global offsets, rather than as user-specific interpolation coefficients, emphasizes the influence of missing ratings. In other words, a user's opinion is formed not only by what he rated, but also by what he did not rate. For example, suppose that a movie ratings dataset shows that users that rate “Lord of the Rings 3” high also gave high ratings to “Lord of the Rings 1–2”. This will establish high weights from “Lord of the Rings 1–2” to “Lord of the Rings 3”. Now, if a user did not rate “Lord of the Rings 1–2” at all, his predicted rating for “Lord of the Rings 3” will be penalized, as some necessary weights cannot be added to the sum.

For prior models ((3),(4)) that interpolated  $r_{ui} - b_{ui}$  from  $\{r_{uj} - b_{uj} | j \in S^k(i;u)\}$ , it was necessary to maintain compatibility between the  $b_{ui}$  values and the  $b_{uj}$  values. However, here we do not use interpolation, so we can decouple the definitions of  $b_{ui}$  and  $b_{uj}$ . Accordingly, a more general prediction rule would be:  $\hat{r}_{ui} = \tilde{b}_{ui} + \sum_{j \in R(u)} (r_{uj} - b_{uj})w_{ij} + \sum_{j \in N(u)} c_{ij}$ . The constant  $\tilde{b}_{ui}$  can represent predictions of  $r_{ui}$  by other methods such as a latent factor model. Here, we suggest the following rule that was found to work well:

$$\hat{r}_{ui} = \mu + b_u + b_i + \sum_{j \in R(u)} (r_{uj} - b_{uj})w_{ij} + \sum_{j \in N(u)} c_{ij} \quad (7)$$

Importantly, the  $b_{uj}$ 's remain constants, which are derived as explained in Sec. 2.1. However, the  $b_u$ 's and  $b_i$ 's become parameters, which are optimized much like the  $w_{ij}$ 's and  $c_{ij}$ 's.

We have found that it is beneficial to normalize sums in the model leading to the form:

$$\begin{aligned} \hat{r}_{ui} = & \mu + b_u + b_i + |R(u)|^{-\alpha} \sum_{j \in R(u)} (r_{uj} - b_{uj})w_{ij} \\ & + |N(u)|^{-\alpha} \sum_{j \in N(u)} c_{ij} \end{aligned} \quad (8)$$

The constant  $\alpha$  controls the extent of normalization. A non-normalized rule ( $\alpha = 0$ ), encourages greater deviations from baseline estimates for users that provided many ratings (high  $|R(u)|$ ) or plenty of implicit feedback (high  $|N(u)|$ ). On the other hand, a fully normalized rule, with  $\alpha = 1$ , will eliminate the effect of number of ratings on deviations from baseline estimates. In many cases it would be a good practice for recommender systems to have greater deviation from baselines for users that rate a lot. This way, we take more risk with well modeled users that provided much input. For such users we are willing to predict quirker and less common recommendations. At the same time, we are less certain about the modeling of

<sup>3</sup>In many cases it would be reasonable to attach significance weights to implicit feedback. This requires a modification to our formula which, for simplicity, will not be considered here.

users that provided only a little input, in which case we would like to stay with safe estimates close to the baseline values. Our experience with the Netflix dataset shows that best results are achieved with partial normalization, which allows moderately higher deviations for heavy raters. Thus, we set  $\alpha = 0.5$ , as in the prediction rule:

$$\begin{aligned}\hat{r}_{ui} = & \mu + b_u + b_i + |\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}(u)} (r_{uj} - b_{uj}) w_{ij} \\ & + |\mathbf{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{N}(u)} c_{ij}\end{aligned}\quad (9)$$

As an optional refinement, complexity of the model can be reduced by pruning parameters corresponding to unlikely item-item relations. Let us denote by  $S^k(i)$  the set of  $k$  items most similar to  $i$ , as determined by the similarity measure  $s_{ij}$ . Additionally, we use  $\mathbf{R}^k(i; u) \stackrel{\text{def}}{=} \mathbf{R}(u) \cap S^k(i)$  and  $\mathbf{N}^k(i; u) \stackrel{\text{def}}{=} \mathbf{N}(u) \cap S^k(i)$ .<sup>4</sup> Now, when predicting  $r_{ui}$  according to (9), it is expected that the most influential weights will be associated with items similar to  $i$ . Hence, we replace (9) with:

$$\begin{aligned}\hat{r}_{ui} = & \mu + b_u + b_i + |\mathbf{R}^k(i; u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}^k(i; u)} (r_{uj} - b_{uj}) w_{ij} \\ & + |\mathbf{N}^k(i; u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{N}^k(i; u)} c_{ij}\end{aligned}\quad (10)$$

When  $k = \infty$ , rule (10) coincides with (9). However, for other values of  $k$  it offers the potential to significantly reduce the number of variables involved.

This is our final prediction rule, which allows fast online prediction. More computational work is needed at a pre-processing stage where parameters are estimated. A major design goal of the new neighborhood model was facilitating an efficient global optimization procedure, which prior neighborhood models lacked. Thus, model parameters are learnt by solving the regularized least squares problem associated with (10):

$$\begin{aligned}\min_{b_*, w_*, c_*} \sum_{(u, i) \in \mathcal{K}} & \left( r_{ui} - \mu - b_u - b_i - |\mathbf{N}^k(i; u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{N}^k(i; u)} c_{ij} \right. \\ & \left. - |\mathbf{R}^k(i; u)|^{-\frac{1}{2}} \sum_{j \in \mathbf{R}^k(i; u)} (r_{uj} - b_{uj}) w_{ij} \right)^2 \\ & + \lambda_5 \left( b_u^2 + b_i^2 + \sum_{j \in \mathbf{R}^k(i; u)} w_{ij}^2 + \sum_{j \in \mathbf{N}^k(i; u)} c_{ij}^2 \right)\end{aligned}\quad (11)$$

An optimal solution of this convex problem can be obtained by least square solvers, which are part of standard linear algebra packages. However, we have found that the following simple gradient descent solver works much faster. Let us denote the prediction error,  $r_{ui} - \hat{r}_{ui}$ , by  $e_{ui}$ . We loop through all known ratings in  $\mathcal{K}$ . For a given training case  $r_{ui}$ , we modify the parameters by moving in the opposite direction of the gradient, yielding:

<sup>4</sup>Notational clarification: With other neighborhood models it was beneficial to use  $S^k(i; u)$ , which denotes the  $k$  items most similar to  $i$  among those rated by  $u$ . Hence, if  $u$  rated at least  $k$  items, we will always have  $|S^k(i; u)| = k$ , regardless of how similar those items are to  $i$ . However,  $|\mathbf{R}^k(i; u)|$  is typically smaller than  $k$ , as some of those items most similar to  $i$  were not rated by  $u$ .

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_5 \cdot b_i)$
- $\forall j \in \mathbf{R}^k(i; u) :$   
 $w_{ij} \leftarrow w_{ij} + \gamma \cdot (|\mathbf{R}^k(i; u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_5 \cdot w_{ij})$
- $\forall j \in \mathbf{N}^k(i; u) :$   
 $c_{ij} \leftarrow c_{ij} + \gamma \cdot (|\mathbf{N}^k(i; u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_5 \cdot c_{ij})$

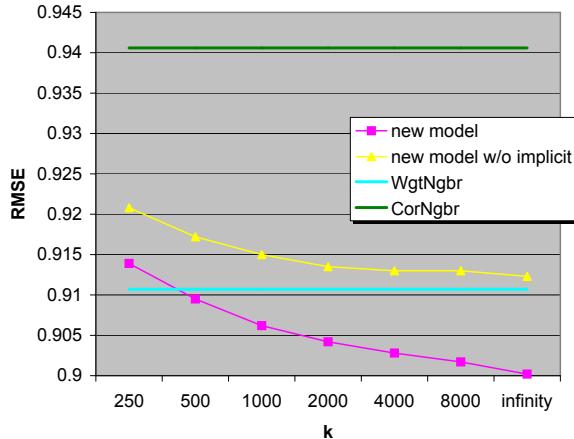
The meta-parameters  $\gamma$  (step size) and  $\lambda_5$  are determined by cross-validation. We used  $\gamma = 0.005$  and  $\lambda_5 = 0.002$  for the Netflix data. Another important parameter is  $k$ , which controls the neighborhood size. Our experience shows that increasing  $k$  always benefits the accuracy of the results on the test set. Hence, the choice of  $k$  should reflect a tradeoff between prediction accuracy and computational cost. In the next section we will describe a factored version of the model, which cancels this tradeoff by allowing us to work with the most accurate  $k = \infty$  while lowering running time.

A typical number of iterations throughout the training data is 15–20. As for time complexity per iteration, let us analyze the most accurate case where  $k = \infty$ , which is equivalent to using prediction rule (9). For each user  $u$  and item  $i \in \mathbf{R}(u)$  we need to modify both  $\{w_{ij} | j \in \mathbf{R}(u)\}$  and  $\{c_{ij} | j \in \mathbf{N}(u)\}$ . Thus the overall time complexity of the training phase is  $O(\sum_u |\mathbf{R}(u)|(|\mathbf{R}(u)| + |\mathbf{N}(u)|))$ .

Experimental results on the Netflix data with the new neighborhood model are presented in Fig. 1. We studied the model under different values of parameter  $k$ . The pink curve shows that accuracy monotonically improves with rising  $k$  values, as root mean squared error (RMSE) falls from 0.9139 for  $k = 250$  to 0.9002 for  $k = \infty$ . (Notice that since the Netflix data contains 17,770 movies,  $k = \infty$  is equivalent to  $k = 17,770$ , where all item-item relations are explored.) We repeated the experiments without using the implicit feedback, that is, dropping the  $c_{ij}$  parameters from our model. The results depicted by the yellow curve show a significant decline in estimation accuracy, which widens as  $k$  grows. This demonstrates the value of incorporating implicit feedback into the model.

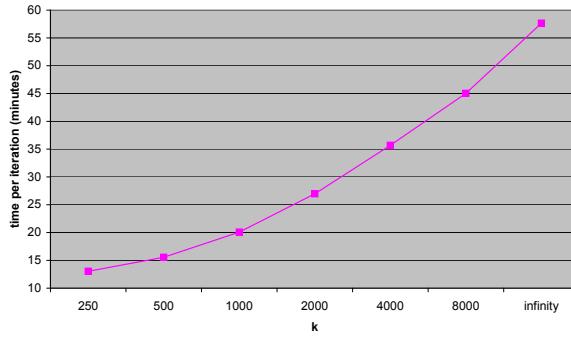
For comparison we provide the results of two previous neighborhood models. First is a correlation-based neighborhood model (following (3)), which is the most popular CF method in the literature. We denote this model as CorNgbr. Second is a newer model [3] that follows (4), which will be denoted as WgtNgbr. For both these two models, we tried to pick optimal parameters and neighborhood sizes, which were 20 for CorNgbr, and 50 for WgtNgbr. The results are depicted by the green and cyan lines. It is clear that the popular CorNgbr method is noticeably less accurate than the other neighborhood models, though its 0.9406 RMSE is still better than the published Netflix's Cinematch RMSE of 0.9514. On the opposite side, our new model delivers more accurate results even when compared with WgtNgbr, as long as the value of  $k$  is at least 500. Notice that the  $k$  value (the  $x$ -axis) is irrelevant to previous models, as their different notion of neighborhood makes neighborhood sizes incompatible. Yet, we observed that while the performance of our algorithm keeps improving as more neighbors are added, this was not true with previous models. For those past models performance peaks with a relatively small number of neighbors and then declines. This may be explained by the fact that in our model parameters are directly learnt from the data through a formal optimization procedure, what facilitates using many more parameters effectively.

Finally, let us consider running time. Previous neighborhood models require very light pre-processing, though, WgtNgbr [3] requires solving a small system of equations for each provided prediction. The new model does involve pre-processing where parameters are estimated. However, online prediction is immediate by



**Figure 1: Comparison of neighborhood-based models.** We measure the accuracy of the new model with and without implicit feedback. Accuracy is measured by RMSE on the Netflix test set, so lower values indicate better performance. RMSE is shown as a function of varying values of  $k$ , which dictates the neighborhood size. For reference, we present the accuracy of two prior models as two horizontal lines: the green line represents a popular method using Pearson correlations, and the cyan line represents a more recent neighborhood model.

following rule (10). Pre-processing time grows with the value of  $k$ . Typical running times per iteration on the Netflix data, as measured on a single processor 3.4GHz Pentium 4 PC, are shown in Fig. 2.



**Figure 2: Running times (minutes) per iteration of the neighborhood model, as a function of the parameter  $k$ .**

#### 4. A FACTORIZED NEIGHBORHOOD MODEL

In the previous section we presented our most accurate neighborhood model, which is based on prediction rule (9) with training time complexity  $O(\sum_u |\mathcal{R}(u)|(|\mathcal{R}(u)| + |\mathcal{N}(u)|))$  and space complexity  $O(m + n^2)$ . (Recall that  $m$  is the number of users, and  $n$  is the number of items.) We could improve time and space complexity by sparsifying the model through pruning unlikely item-item relations. Sparsification was controlled by the parameter  $k \leq n$ , which reduced running time and allowed space complexity of  $O(m + nk)$ . However, as  $k$  gets lower, the accuracy of the model declines as well. Now, we will show how to retain the accuracy of the full dense prediction rule (9), while significantly lowering time and space complexity.

We will factor item-item relationships by associating each item  $i$

with three vectors:  $q_i, x_i, y_i \in \mathbb{R}^f$ . This way, we will confine  $w_{ij}$  to be  $q_i^T x_i$ . Similarly, we impose  $c_{ij} = q_i^T y_j$ . Essentially, these vectors strive to map items into a latent factor space where they are measured against various aspects that are revealed automatically by learning the data. By substituting this into (9) we get the following prediction rule:

$$\begin{aligned} \hat{r}_{ui} = & \mu + b_u + b_i + |\mathcal{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}(u)} (r_{uj} - b_{uj}) q_i^T x_j \\ & + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} q_i^T y_j \end{aligned} \quad (12)$$

Computational gains become more obvious by using the equivalent rule:

$$\begin{aligned} \hat{r}_{ui} = & \mu + b_u + b_i + q_i^T \left( |\mathcal{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}(u)} (r_{uj} - b_{uj}) x_j \right. \\ & \left. + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j \right) \end{aligned} \quad (13)$$

Notice that the bulk of the rule –  $\left( |\mathcal{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}(u)} (r_{uj} - b_{uj}) x_j + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j \right)$  – depends only on  $u$  while being independent of  $i$ . This leads to an efficient way to learn the model parameters. As usual, we minimize the regularized squared error function associated with (12):

$$\begin{aligned} & \min_{q_*, x_*, y_*, b_*} \sum_{(u,i) \in \mathcal{K}} \left( r_{ui} - \mu - b_u - b_i \right. \\ & \left. - q_i^T \left( |\mathcal{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}(u)} (r_{uj} - b_{uj}) x_j + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j \right) \right)^2 \\ & + \lambda_6 \left( b_u^2 + b_i^2 + \|q_i\|^2 + \sum_{j \in \mathcal{R}(u)} \|x_j\|^2 + \sum_{j \in \mathcal{N}(u)} \|y_j\|^2 \right) \end{aligned} \quad (14)$$

We employ a simple gradient descent scheme, which is described in the following pseudo code:

```

LearnFactorizedNeighborhoodModel(Known ratings:  $r_{ui}$ , rank:  $f$ )
% For each item  $i$  compute  $q_i, x_i, y_i \in \mathbb{R}^f$ 
% which form a neighborhood model
Const #Iterations = 20,  $\gamma = 0.002$ ,  $\lambda = 0.04$ 
% Gradient descent sweeps:
for count = 1, ..., #Iterations do
    for  $u = 1, \dots, m$  do
        % Compute the component independent of  $i$ :
         $p_u \leftarrow |\mathcal{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}(u)} (r_{uj} - b_{uj}) x_j$ 
         $p_u \leftarrow p_u + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j$ 
        sum  $\leftarrow 0$ 
        for all  $i \in \mathcal{R}(u)$  do
             $\hat{r}_{ui} \leftarrow \mu + b_u + b_i + q_i^T p_u$ 
             $e_{ui} \leftarrow r_{ui} - \hat{r}_{ui}$ 
            % Accumulate information for gradient steps on  $x_i, y_i$ :
            sum  $\leftarrow$  sum +  $e_{ui} \cdot q_i$ 
            % Perform gradient step on  $q_i, b_u, b_i$ :
             $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$ 
             $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda \cdot b_u)$ 
             $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda \cdot b_i)$ 
        for all  $i \in \mathcal{R}(u)$  do
            % Perform gradient step on  $x_i$ :
             $x_i \leftarrow x_i + \gamma \cdot (|\mathcal{R}(u)|^{-\frac{1}{2}} \cdot (r_{ui} - b_{ui}) \cdot \text{sum} - \lambda \cdot x_i)$ 
        for all  $i \in \mathcal{N}(u)$  do
            % Perform gradient step on  $y_i$ :
             $y_i \leftarrow y_i + \gamma \cdot (|\mathcal{N}(u)|^{-\frac{1}{2}} \cdot \text{sum} - \lambda \cdot y_i)$ 
    return  $\{q_i, x_i, y_i | i = 1, \dots, n\}$ 

```

| #factors       | 50      | 100    | 200    | 500    |
|----------------|---------|--------|--------|--------|
| RMSE           | 0.9037  | 0.9013 | 0.9000 | 0.8998 |
| time/iteration | 4.5 min | 8 min  | 14 min | 34 min |

**Table 1: Performance of the factorized item-item neighborhood model.** The models with  $\geq 200$  factors slightly outperform the non-factorized model, while providing much shorter running time.

tion technique will inevitably degrade accuracy by missing important relations, as demonstrated in the previous section. In addition, identification of the top  $k$  most similar items in such a high dimensional space is a non-trivial task that can require considerable computational efforts. All these issues do not exist in our factorized neighborhood model, which offers a linear time and space complexity without trading off accuracy.

**Discussion** The factorized neighborhood model resembles principles of some latent factor models. The important distinction is that here we factorize the item-item relationships, rather than the ratings themselves. The results reported in Table 1 are comparable to those of the widely used SVD model, which generates results with RMSEs ranging in 0.9046–0.9009 for 50–200 factors [17]. However, some sophisticated extensions of the SVD model would result in even better accuracy [17]. Nonetheless, the factorized neighborhood model retains the practical advantages of traditional neighborhood models, which are difficult to achieve with latent factor models. Here, we refer to the ability to explain recommendations and to immediately reflect new ratings, as discussed in Sec. 2.2.

As a side remark, we would like to mention that the decision to use three separate sets of factors was intended to give us more flexibility. Indeed, on the Netflix data this allowed achieving most accurate results. However, another reasonable choice could be using a smaller set of vectors, e.g., by requiring:  $q_i = x_i$  (implying symmetric weights:  $w_{ij} = w_{ji}$ ).

## 4.1 A user-user model

A user-user neighborhood model predicts ratings by considering how like-minded users rated the same items. Such models can be implemented by switching the roles of users and items throughout our derivation of the item-item model. Here, we would like to concentrate on a user-user model, which is analogous to the item-item model of (9). The major difference is replacing the  $w_{ij}$  weights relating item pairs, with weights relating user pairs:

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathcal{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathcal{R}(i)} (r_{vj} - b_{vj}) w_{uv} \quad (15)$$

The set  $\mathcal{R}(i)$  contains all the users for who rated item  $i$ . Notice that here we decided to not account for implicit feedback. This is because adding such feedback was not very beneficial for the user-user model when working with the Netflix data.

User-user models can become useful in various situations. For example, some recommenders may deal with items that are rapidly replaced, thus making item-item relations very volatile. On the other hand, the user base is pretty stable enabling to establish long term relations between users. An example of such a case is a recommender system for web articles or news items which are rapidly changing by their nature; see, e.g., [9]. In such cases, systems centered around user-user relations are more appealing.

In addition, user-user approaches identify different kinds of relations that item-item approaches may fail to recognize, and thus can be useful on certain occasions. For example, suppose that we

The time complexity of this model is linear with the input size  $O(f \cdot \sum_u (|\mathcal{R}(u)| + |\mathcal{N}(u)|))$ , which is significantly better than the non-factorized model that required time  $O(\sum_u |\mathcal{R}(u)|(|\mathcal{R}(u)| + |\mathcal{N}(u)|))$ . We measured the performance of the model on the Netflix data; see Table 1. Accuracy is improved as we use more factors (increasing  $f$ ). However, going beyond 200 factors could barely improve performance, while slowing running time. Interestingly, we have found that with  $f \geq 200$  accuracy negligibly exceeds that of the best non-factorized model (with  $k = \infty$ ). In addition, the improved time complexity translates into a big difference in wall-clock measured running time. For example, the time-per-iteration for the non-factorized model (with  $k = \infty$ ) was close to 58 minutes. On the other hand, a factorized model with  $f = 200$  could complete an iteration in 14 minutes without degrading accuracy at all.

The most important benefit of the factorized model is the reduced space complexity, which is  $O(m + nf)$  – linear in the input size. Previous neighborhood models required storing all pairwise relations between items, leading to a quadratic space complexity of  $O(m + n^2)$ . For the Netflix dataset which contains 17,770 movies, such quadratic space can still fit within core memory. Some commercial recommenders process a much higher number of items. For example, a leading online movie rental service is currently advertising offering over 100,000 titles. Music download shops offer even more titles. Such more comprehensive systems with data on 100,000s items eventually need to resort to external storage in order to fit the entire set of pairwise relations. However, as number of items is growing towards millions, as in the Amazon item-item recommender system, which accesses stored similarity information for several million catalog items [18], designers must keep a sparsified version of the pairwise relations. To this end, only values relating an item to its top- $k$  most similar neighbors are stored thereby reducing space complexity to  $O(m + nk)$ . However, a sparsifica-

| #factors       | 50     | 100    | 200     | 500    |
|----------------|--------|--------|---------|--------|
| RMSE           | 0.9119 | 0.9110 | 0.9101  | 0.9093 |
| time/iteration | 3 min  | 5 min  | 8.5 min | 18 min |

**Table 2: Performance of the factorized user-user neighborhood model.**

want to predict  $r_{ui}$ , but none of the items rated by user  $u$  is really relevant to  $i$ . In this case, an item-item approach will face obvious difficulties. However, when employing a user-user perspective, we may find a set of users similar to  $u$ , who rated  $i$ . The ratings of  $i$  by these users would allow us to improve prediction of  $r_{ui}$ .

The major disadvantage of user-user model is computational. Since typically there are many more users than items, precomputing and storing all user-user relations, or even a reasonably sparsified version thereof, is overly expensive or completely impractical. In addition to the high  $O(m^2)$  space complexity, the time complexity for optimizing model (15) is also much higher than its item-item counterpart, being  $O(\sum_i |\mathcal{R}(i)|^2)$  (notice that  $|\mathcal{R}(i)|$  is expected to be much higher than  $|\mathcal{R}(u)|$ ). These issues render user-user models as a less practical choice.

**A factorized model** All those computational differences disappear by factorizing the user-user model along the same lines as in the item-item model. Now, we associate each user  $u$  with two vector  $p_u, z_u \in \mathbb{R}^f$ . We assume the user-user relations to satisfy:  $w_{uv} = p_u^T z_v$ . Let us substitute this into (15) to get:

$$\hat{r}_{ui} = \mu + b_u + b_i + |\mathcal{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathcal{R}(i)} (r_{vj} - b_{vj}) p_u^T z_v \quad (16)$$

Once again, an efficient computation is achieved by including the terms that depends on  $i$  but are independent of  $u$  in a separate sum, so the prediction rule is presented in the equivalent form:

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T |\mathcal{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathcal{R}(i)} (r_{vj} - b_{vj}) z_v \quad (17)$$

In a parallel fashion to the item-item model, all parameters are learnt in linear time  $O(f \cdot \sum_i |\mathcal{R}(i)|)$ . The space complexity is also linear with the input size being  $O(n + mf)$ . This significantly lowers the complexity of the user-user model compared to previously known results. In fact, running time measured on the Netflix data shows that now the user-user model is even faster than the item-item model; see Table 2. We should remark that unlike the item-item model, our implementation of the user-user model did not account for implicit feedback, what probably lead to its shorter running time. Accuracy of the user-user model is significantly better than that of the widely-used correlation-based item-item models, which achieves RMSE=0.9406 as reported in Sec. 3. Furthermore, accuracy is slightly better than the variant of the item-item model that also did not account for implicit feedback (yellow curve in Fig. 1). This is quite surprising given the common wisdom that item-item methods are more accurate than user-user ones. It appears that a well implemented user-user model can match speed and accuracy of an item-item model. However, our item-item model could significantly benefit by accounting for implicit feedback.

**Fusing item-item and user-user models** Since item-item and user-user models address different aspects of the data, overall accuracy is expected to improve by combining predictions of both models. Such an approach was previously suggested and was shown to improve accuracy; see, e.g. [5, 29]. However, past efforts were based on blending the item-item and user-user predictions during a post-processing stage, after each individual model is separately trained independently of the other model. A more principled ap-

proach will optimize the two models simultaneously, letting them know of each other while parameters are being learnt. Thus, throughout the entire training phase each model is aware of the capabilities of the other model and strives to complement it. Our approach, which states the neighborhood models as formal optimization problems, allows doing that naturally. We devise a model which sums the item-item model (12) and the user-user model (16), leading to:

$$\begin{aligned} \hat{r}_{ui} = & \mu + b_u + b_i + |\mathcal{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}(u)} (r_{uj} - b_{uj}) q_i^T x_i \\ & + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} q_i^T y_j \\ & + |\mathcal{R}(i)|^{-\frac{1}{2}} \sum_{v \in \mathcal{R}(i)} (r_{vj} - b_{vj}) p_u^T z_v \end{aligned} \quad (18)$$

Model parameters are learnt by gradient descent optimization of the associated squared error function. Our experiments with the Netflix data show that prediction accuracy is indeed better than that of each individual model. For example, with 100 factors the obtained RMSE is 0.8966, while with 200 factors the obtained RMSE is 0.8953.

Here we would like to comment that our approach allows integrating the neighborhood models also with completely different models in a similar way. For example, in [17] we showed an integrated model that combines the item-item model with a latent factor model (called SVD++), thereby achieving improved prediction accuracy with RMSE below 0.887. Therefore, other possibilities with potentially better accuracy should be explored before considering the integration of an item-item and user-user models.

## 5. EVALUATION THROUGH A TOP-K RECOMMENDER

So far, we have followed a common practice with the Netflix dataset to evaluate prediction accuracy by the RMSE measure. Achievable RMSE values on the Netflix test data lie in a quite narrow range. A simple prediction rule, which estimates  $r_{ui}$  as the mean rating of movie  $i$ , will result in RMSE=1.053. Notice that this rule represents a sensible “best sellers list” approach, where the same recommendation applies to all users. By applying personalization, more accurate predictions are obtained. This way, Netflix Cinematch system could achieve a RMSE of 0.9514. In this paper, we suggested methods that lower the RMSE to below 0.9. Successful improvements of recommendation quality depend on achieving the elusive goal of enhancing users’ satisfaction. Thus, a crucial question is: what effect on user experience should we expect by lowering the RMSE by, say, 5%? For example, is it possible that a solution with a slightly better RMSE will lead to completely different and better recommendations? This is central to justifying research on accuracy improvements in recommender systems. We would like to shed some light on the issue, by examining the effect of lowered RMSE on a practical situation.

A common case facing recommender systems is providing “top K recommendations”. That is, the system needs to suggest the top K products to a user. For example, recommending the user a few specific movies which are supposed to be most appealing to him. We would like to investigate the effect of lowering the RMSE on the quality of top K recommendations. Somewhat surprisingly, the Netflix dataset can be used to evaluate this.

Recall that in addition to the test set, Netflix also provided a validation set for which the true ratings are published. We used all 5-star ratings from the validation set as a proxy for movies that

interest users.<sup>5</sup> Our goal is to find the relative place of these “interesting movies” within the total order of movies sorted by predicted ratings for a specific user. To this end, for each such movie  $i$ , rated 5-stars by user  $u$ , we select 1000 additional random movies and predict the ratings by  $u$  for  $i$  and for the other 1000 movies. Finally, we order the 1001 movies based on their predicted rating, in a decreasing order. This simulates a situation where the system needs to recommend movies out of 1001 available ones. Thus, those movies with the highest predictions will be recommended to user  $u$ . Notice that the 1000 movies are random, some of which may be of interest to user  $u$ , but most of them are probably of no interest to  $u$ . Hence, the best hoped result is that  $i$  (for which we know  $u$  gave the highest rating of 5) will precede the rest 1000 random movies, thereby improving the appeal of a top-K recommender. There are 1001 different possible ranks for  $i$ , ranging from the best case where none (0%) of the random movies appears before  $i$ , to the worst case where all (100%) of the random movies appear before  $i$  in the sorted order. Overall, the validation set contains 384,573 5-star ratings. For each of them (separately) we draw 1000 random movies, predict associated ratings, and derive a ranking between 0% to 100%. Then, the distribution of the 384,573 ranks is analyzed. (Remark: since the number 1000 is arbitrary, reported results are in percentiles (0%–100%), rather than in absolute ranks (0–1000).)

We used this methodology to evaluate five different methods. The first method is the aforementioned non-personalized prediction rule, which employs movie means to yield RMSE=1.053. Henceforth, it will be denoted as MovieAvg. The second method is a correlation-based neighborhood model, which is the most popular approach in the CF literature. As mentioned in Sec. 3, it achieves a RMSE of 0.9406 on the test set, and was named CorNgbr. The third method is the improved neighborhood approach of [3], which we named WgtNgbr and could achieve RMSE= 0.9107 on the test set. Fourth is the SVD latent factor model, with 100 factors thereby achieving RMSE=0.9025 (see [17]). Finally, we consider our new neighborhood model with a full set of neighbors ( $k = \infty$ ), achieving RMSE=0.9002.

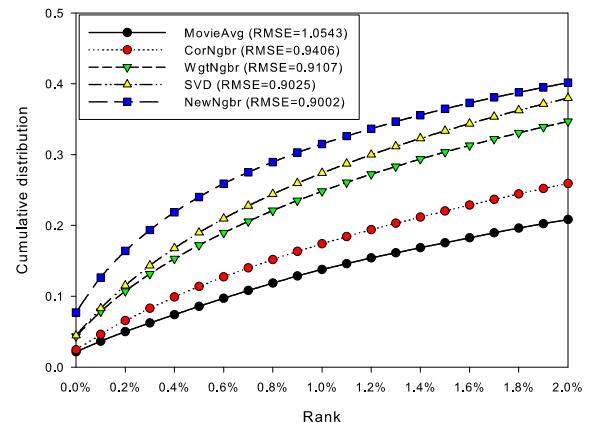
Figure 3 plots the cumulative distribution of the computed percentile ranks for the five methods over the 384,573 evaluated cases. In order to get into the top-K recommendations, a product should be ranked before almost all others. For example, if 600 products are considered, and three of them will be suggested to the user, only those ranked 0.5% or lower are relevant. In a sense, there is no difference between placing a desired 5-star movie at the top 5%, top 20% or top 80%, as none of them is good enough to be presented to the user. Accordingly, we concentrate on cumulative ranks distribution between 0% and 2% (top 20 ranked items out of 1000).

As the figure shows, there are very significant differences among the methods. For example, the new neighborhood (NewNgbr) method has a probability of 0.077 to place a 5-star movie before all other 1000 movies (rank=0%). This is more than 3.5 times better than the chance of the MovieAvg method to achieve the same. In addition, it is over three times better than the chance of the popular CorNgbr to achieve the same. The other two methods, WgtNgbr and SVD, have a probability of around 0.043 to achieve the same. The practical interpretation is that if about 0.1% of the items are selected to be suggested to the user, the new neighborhood method has a significantly higher chance to pick a specified 5-star rated movie. Similarly, NewNgbr has a probability of 0.163 to place the 5-star movie before at least 99.8% of the random movies ( $\text{rank} \leq 0.2\%$ ). For com-

parison, MovieAvg and CorNgbr have much slimmer chances of achieving the same: 0.050 and 0.065, respectively. The remaining two methods, WgtNgbr and SVD, lie between with probabilities of 0.107 and 0.115, respectively. Thus, if one movie out of 500 is to be suggested, its probability of being a specific 5-star rated one becomes noticeably higher with the new neighborhood model.

We are encouraged, even somewhat surprised, by the results. It is evident that small improvements in RMSE translate into significant improvements in quality of the top K products. In fact, based on RMSE differences, we did not expect the new neighborhood model to deliver such an emphasized improvement in the test. Similarly, we did not expect the very weak performance of the popular correlation based neighborhood scheme, which could not improve much upon a non-personalized scheme.

Notice that our evaluation answers the question: given a user and a 5-star rated movie, what is the chance that a particular method will rank this movie within the top X%? One can come up with other related questions, which we did not analyze here. For example, we can move from a single specific 5-star movie, to all movies in which the user is interested, thus answering a question like: given a user and set of all movies she would rate 5 stars, what is the chance that a method will rank at least one 5-star rated movie within the top X%? However, since the validation set provides us with a very limited view of user preferences, we do not really know the full set of movies that she likes, and hence could not evaluate performance regarding the latter question.



**Figure 3: Comparing the performance of five methods on a top-K recommendation task, where a few products need to be suggested to a user. Values on the  $x$ -axis stand for the percentile ranking of a 5-star rated movie; lower values represent more successful recommendations. We experiment with 384,573 cases and show the cumulative distribution of the results. The plot concentrates on the more relevant region, pertaining to low  $x$ -axis values. The new neighborhood method has the highest probability of placing a 5-star rated movie within the top-K recommendations. On the other hand, the non-personalized MovieAvg method and the popular correlation-based neighborhood method (CorNgbr) achieve the lowest probabilities.**

## 6. CONCLUSIONS

This work proposed a new neighborhood based model, which unlike previous neighborhood methods is based on formally optimizing a global cost function. This leads to improved prediction

<sup>5</sup>In this study, the validation set is excluded from the training data.

accuracy, while maintaining merits of the neighborhood approach such as explainability of predictions and ability to handle new ratings (or new users) without re-training the model.

In addition we suggested a factorized version of the neighborhood model, which improves its computational complexity while retaining prediction accuracy. In particular, it is free from the quadratic storage requirements that limited past neighborhood models. Thus, the new method offers scalability to a very large number of items without resorting to complicated sparsification techniques that tend to lower accuracy. The improved complexity of the factorized model is especially emphasized with the user-user models, which were considered so far as computationally inefficient. Now they become as efficient as the item-item ones, thus making them a viable option. In fact, thanks to their vastly improved efficiency, we could build full user-user models that achieve accuracy competitive with item-item models.

Quality of a recommender system is expressed through multiple dimensions including: accuracy, diversity, ability to surprise with unexpected recommendations, explainability, appropriate top-K recommendations, and computational efficiency. Some of those criteria are relatively easy to measure, such as accuracy and efficiency that were addressed in this work. Some other aspects are more elusive and harder to quantify. We suggested a novel approach for measuring the success of a top-K recommender, which is central to most systems where a few products should be suggested to each user. It is notable that evaluating top-K recommenders significantly sharpens the differences between the methods, beyond what a traditional accuracy measure could show.

## 7. REFERENCES

- [1] G. Adomavicius and A. Tuzhilin, “Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions”, *IEEE Transactions on Knowledge and Data Engineering* **17** (2005), 634–749.
- [2] K. Ali and W. van Stam, “TiVo: Making Show Recommendations Using a Distributed Collaborative Filtering Architecture”, *Proc. 10th ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining*, pp. 394–401, 2004.
- [3] R. Bell and Y. Koren, “Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights”, *IEEE International Conference on Data Mining (ICDM’07)*, pp. 43–52, 2007.
- [4] R. Bell and Y. Koren, “Lessons from the Netflix Prize Challenge”, *SIGKDD Explorations* **9** (2007), 75–79.
- [5] R. M. Bell, Y. Koren and C. Volinsky, “Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems”, *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.
- [6] J. Bennet and S. Lanning, “The Netflix Prize”, *KDD Cup and Workshop*, 2007. [www.netflixprize.com](http://www.netflixprize.com).
- [7] J. Canny, “Collaborative Filtering with Privacy via Factor Analysis”, *Proc. 25th ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR02)*, pp. 238–245, 2002.
- [8] D. Blei, A. Ng, and M. Jordan, “Latent Dirichlet Allocation”, *Journal of Machine Learning Research* **3** (2003), 993–1022.
- [9] A. Das, M. Datar, A. Garg and S. Rajaram, “Google News Personalization: Scalable Online Collaborative Filtering”, *WWW07*, pp. 271–280, 2007.
- [10] S. Deerwester, S. Dumais, G. W. Furnas, T. K. Landauer and R. Harshman, “Indexing by Latent Semantic Analysis”, *Journal of the Society for Information Science* **41** (1990), 391–407.
- [11] S. Funk, “Netflix Update: Try This At Home”, <http://sifter.org/~simon/journal/20061211.html>, 2006.
- [12] D. Goldberg, D. Nichols, B. M. Oki and D. Terry, “Using Collaborative Filtering to Weave an Information Tapestry”, *Communications of the ACM* **35** (1992), 61–70.
- [13] J. L. Herlocker, J. A. Konstan and J. Riedl, , “Explaining Collaborative Filtering Recommendations”, *Proc. ACM conference on Computer Supported Cooperative Work*, pp. 241–250, 2000.
- [14] J. L. Herlocker, J. A. Konstan, A. Borchers and John Riedl, “An Algorithmic Framework for Performing Collaborative Filtering”, *Proc. 22nd ACM SIGIR Conference on Information Retrieval*, pp. 230–237, 1999.
- [15] T. Hofmann, “Latent Semantic Models for Collaborative Filtering”, *ACM Transactions on Information Systems* **22** (2004), 89–115.
- [16] D. Kim and B. Yum, “Collaborative Filtering Based on Iterative Principal Component Analysis”, *Expert Systems with Applications* **28** (2005), 823–830.
- [17] Y. Koren, “Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model”, *Proc. 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.
- [18] G. Linden, B. Smith and J. York, “Amazon.com Recommendations: Item-to-item Collaborative Filtering”, *IEEE Internet Computing* **7** (2003), 76–80.
- [19] Benjamin M. Marlin, Richard S. Zemel, Sam Roweis, and Malcolm Slaney, “Collaborative filtering and the Missing at Random Assumption”, *Proc. 23rd Conference on Uncertainty in Artificial Intelligence*, 2007.
- [20] D.W. Oard and J. Kim, “Implicit Feedback for Recommender Systems”, *Proc. 5th DELOS Workshop on Filtering and Collaborative Filtering*, pp. 31–36, 1998.
- [21] S.-T. Park and D. M. Pennock, “Applying Collaborative Filtering Techniques to Movie Search for Better Ranking and Browsing”, *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 550559, 2007.
- [22] A. Paterek, “Improving Regularized Singular Value Decomposition for Collaborative Filtering”, *Proc. KDD Cup and Workshop*, 2007.
- [23] R. Salakhutdinov, A. Mnih and G. Hinton, “Restricted Boltzmann Machines for Collaborative Filtering”, *Proc. 24th Annual International Conference on Machine Learning*, pp. 791–798, 2007.
- [24] R. Salakhutdinov and A. Mnih, “Probabilistic Matrix Factorization”, *Advances in Neural Information Processing Systems 20 (NIPS’07)*, pp. 1257–1264, 2008.
- [25] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, “Application of Dimensionality Reduction in Recommender System – A Case Study”, *WEBKDD’2000*.
- [26] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, “Item-based Collaborative Filtering Recommendation Algorithms”, *Proc. 10th International Conference on the World Wide Web*, pp. 285–295, 2001.
- [27] G. Takacs, I. Pilaszy, B. Nemeth and D. Tikk, “Major Components of the Gravity Recommendation System”,

- SIGKDD Explorations* **9** (2007), 80–84.
- [28] N. Tintarev and J. Masthoff, “A Survey of Explanations in Recommender Systems”, *ICDE'07 Workshop on Recommender Systems and Intelligent User Interfaces*, 2007.
  - [29] J. Wang, A. P. de Vries and M. J. T. Reinders, “Unifying User-based and Item-based Collaborative Filtering Approaches by Similarity Fusion”, *Proc. 29th ACM SIGIR Conference on Information Retrieval*, pp. 501–508, 2006.

# A Survey of Accuracy Evaluation Metrics of Recommendation Tasks

**Asela Gunawardana**

*Microsoft Research*

*One Microsoft Way*

*Redmond, WA 98052-6399, USA*

ASELAG@MICROSOFT.COM

**Guy Shani**

*Information Systems Engineering*

*Ben Gurion University*

*Beer Sheva, Israel*

SHANIGU@BGU.AC.IL

**Editor:** Lyle Ungar

## Abstract

*Recommender systems* are now popular both commercially and in the research community, where many algorithms have been suggested for providing recommendations. These algorithms typically perform differently in various domains and tasks. Therefore, it is important from the research perspective, as well as from a practical view, to be able to decide on an algorithm that matches the domain and the task of interest. The standard way to make such decisions is by comparing a number of algorithms offline using some evaluation metric. Indeed, many evaluation metrics have been suggested for comparing recommendation algorithms. The decision on the proper evaluation metric is often critical, as each metric may favor a different algorithm. In this paper we review the proper construction of offline experiments for deciding on the most appropriate algorithm. We discuss three important tasks of recommender systems, and classify a set of appropriate well known evaluation metrics for each task. We demonstrate how using an improper evaluation metric can lead to the selection of an improper algorithm for the task of interest. We also discuss other important considerations when designing offline experiments.

**Keywords:** recommender systems, collaborative filtering, statistical analysis, comparative studies

## 1. Introduction

Recommender systems can now be found in many modern applications that expose the user to a huge collections of items. Such systems typically provide the user with a list of recommended items they might prefer, or supply guesses of how much the user might prefer each item. These systems help users to decide on appropriate items, and ease the task of finding preferred items in the collection.

For example, the DVD rental provider Netflix<sup>1</sup> displays predicted ratings for every displayed movie in order to help the user decide which movie to rent. The online book retailer Amazon<sup>2</sup> provides average user ratings for displayed books, and a list of other books that are bought by users who buy a specific book. Microsoft provides many free downloads for users, such as bug fixes, products and so forth. When a user downloads some software, the system presents a list

---

1. This can be found at [www.netflix.com](http://www.netflix.com).

2. This can be found at [www.amazon.com](http://www.amazon.com).

of additional items that are downloaded together. All these systems are typically categorized as recommender systems, even though they provide diverse services.

In the past decade, there has been a vast amount of research in the field of recommender systems, mostly focusing on designing new algorithms for recommendations. An application designer who wishes to add a recommendation system to her application has a large variety of algorithms at her disposal, and must make a decision about the most appropriate algorithm for her application. Typically, such decisions are based on offline experiments, comparing the performance of a number of candidate algorithms over real data. The designer can then select the best performing algorithm, given structural constraints. Furthermore, most researchers who suggest new recommendation algorithms also compare the performance of their new algorithm to a set of existing approaches. Such evaluations are typically performed by applying some evaluation metric that provides a ranking of the candidate algorithms (usually using numeric scores).

Many evaluation metrics have been used to rank recommendation algorithms, some measuring similar features, but some measuring drastically different quantities. For example, methods such as the Root of the Mean Square Error (RMSE) measure the distance between predicted preferences and true preferences over items, while the Recall method computes the portion of favored items that were suggested. Clearly, it is unlikely that a single algorithm would outperform all others over all possible methods.

Therefore, we should expect different metrics to provide different rankings of algorithms. As such, selecting the proper evaluation metric to use has a crucial influence on the selection of the recommender system algorithm that will be selected for deployment. This survey reviews existing evaluation metrics, suggesting an approach for deciding which evaluation metric is most appropriate for a given application.

We categorize previously suggested recommender systems into three major groups, each corresponding to a different *task*. The first obvious task is to recommend a set of good (interesting, useful) items to the user. In this task it is assumed that all good items are interchangeable. A second, less discussed, although highly important task is utility optimization. For example, many e-commerce websites use a recommender system, hoping to increase their revenues. In this case, the task is to present a set of recommendations that will optimize the retailer revenue. Finally, a very common task is the prediction of user opinion (e.g., rating) over a set of items. While this may not be an explicit act of recommendation, much research in recommender systems focuses on this task, and so we address it here.

For each such task we review a family of common evaluation metrics that measure the performance of algorithms on that task. We discuss the properties of each such metric, and why it is most appropriate for a given task.

In some cases, applying incorrect evaluation metrics may result in selecting an inappropriate algorithm. We demonstrate this by experimenting with a wide collection of data sets, comparing a number of algorithms using various evaluation metrics, showing that the metrics rank the algorithms differently.

We also discuss the proper design of an offline experiment, explaining how the data should be split, which measurements should be taken, how to determine if differences in performance are statistically significant, and so forth. We also describe a few common pitfalls that may produce results that are not statistically sound.

The paper is structured as follows: we begin with some necessary background on recommender approaches (Section 2). We categorize recommender systems into a set of three tasks in Section 3.

We then discuss evaluation protocols, including online experimentation, offline testing, and statistical significance testing of results in Section 4. We proceed to review a set of existing evaluation metrics, mapping them to the appropriate task (Section 5). We then provide (Section 6) some examples of applying different metrics to a set of algorithms, resulting in questionable rankings of these algorithms when inappropriate measures are used. Following this, we discuss some additional relevant topics that arise (Section 7) and some related work (Section 8), and then conclude (Section 9).

## 2. Algorithmic Approaches

There are two dominant approaches for computing recommendations for the *active user*—the user that is currently interacting with the application and the recommender system. First, the *collaborative filtering* approach (Breese et al., 1998) assumes that users who agreed on preferred items in the past will tend to agree in the future too. Many such methods rely on a matrix of user-item ratings to predict unknown matrix entries, and thus to decide which items to recommend.

A simple approach in this family (Konstan et al., 2006), commonly referred to as *user based collaborative filtering*, identifies a neighborhood of users that are similar to the *active user*. This set of neighbors is based on the similarity of observed preferences between these users and the active user. Then, items that were preferred by users in the neighborhood are recommended to the active user. Another approach (Linden et al., 2003), known as *item based collaborative filtering* recommends items also preferred by users that prefer a particular *active item* to other users that also prefer that active item. In collaborative filtering approaches, the system only has access to the item and user identifiers, and no additional information over items or users is used. For example, websites that present recommendations titled “users who preferred this item also prefer” typically use some type of collaborative filtering algorithm.

A second popular approach is the *content-based* recommendation. In this approach, the system has access to a set of item features. The system then learns the user preferences over features, and uses these computed preferences to recommend new items with similar features. Such recommendations are typically titled “similar items”. User’s features, if available, such as demographics (e.g., gender, age, geographic location) can also provide valuable information.

Each approach has advantages and disadvantages, and a multitude of algorithms from each family, as well as a number of hybrid approaches have been suggested. This paper, though, makes no distinction between the underlying recommendation algorithms when evaluating their performance. Just as users should not need to take into account the details of the underlying algorithm when using the resulting recommendations, it is inappropriate to select different evaluation metrics for different recommendation approaches. In fact, doing so would make it difficult to decide which approach to employ in a particular application.

## 3. Recommender Systems Tasks

Providing a single definition for recommender systems is difficult, mainly because systems with different objectives and behaviors are grouped together under that name. Below, we categorize recommender systems into three classes, based on the recommendation task that they are designed for McNee et al. (2006). In fact, there have been several previous attempts to classify existing recommenders (see, e.g., Montaner et al. 2003 and Schafer et al. 1999). We, however, are interested

in the proper evaluation of such algorithms, and our classification is derived from that goal. While there may be recommender systems that do not fit well into the classes that we suggest, we believe that the vast majority of the recommender systems attempt to achieve one of these tasks, and can thus be classified as we suggest.

### 3.1 Recommending Good Items

Perhaps the most common task of recommendation engines is to recommend good items to users (Herlocker et al., 2004; McNee et al., 2006). Such systems typically present a list of items that the user is predicted to prefer. The user can then select (add to the shopping basket, view, ...) one or more of the suggested items. There are many examples of such systems. In the Amazon website, for instance, when the user is looking at an item, the system presents below a list of other items that the user may be interested in. Another example can be found in Netflix—when a user adds a movie to her queue, the system displays a list of other recommended movies that the user may want to add to the queue too. There are several considerations when creating recommendation lists. We identify below two sub-tasks that comply with different requirements.

#### 3.1.1 RECOMMENDING SOME GOOD ITEMS

In this sub-task, we make the assumption that there is a large number of good items that may appeal to the user, and the user does not have enough resources (time, money) to select all items. In this case we can only present a part of the preferred item set. Thus, it is likely that many preferred items will be missing from the list. In this sub-task, it is more important not to present any disliked item than to find all the good items.

This is typically the case in recommender systems that suggest media items, such as movies, books, or news items. In all these cases the number of alternatives is huge, and the user cannot possibly watch all the recommended movies, or read all the relevant books.

#### 3.1.2 RECOMMENDING ALL GOOD ITEMS

A less popular case is when the system should recommend all important items. Examples of such systems are recommenders that predict which scientific papers should be cited, or legal databases (Herlocker et al., 2004; McNee et al., 2006), where it is important not to overlook any possible case. In this sub-task, the system can present longer lists of items, trying to avoid missing a relevant item.

### 3.2 Optimizing Utility

With the rise of e-commerce websites, another recommendation task became highly important—maximizing the profits of the website. Online retailers are willing to invest in recommender systems hoping to increase their revenue. There are many ways by which a recommender system can increase revenue. The simplest way is through cross-selling; by suggesting additional items to the users, we increase the probability that the user will buy more than he originally intended. In an online news provider, where most revenue comes from display advertisements, the system can increase profit by keeping the users in the website for longer time periods, as the performance of an advertising campaign is often measured in terms of “ $x$ -minute reach,” which is the number of consumers in a particular market that are exposed to the ad for  $x$  minutes. In such cases, it is in the best interest of the system to suggest items in order to lengthen the session. In a subscription

service, where revenue comes from users paying a regular subscription, the goal may be to allow users to easily reach items of interest. In this case, the system should suggest items such that the user reaches items of interest with minimal effort.

The utility function to be optimized can be more complicated, and in particular, may be a function of the entire set of recommendations and their presentation to the user. For example, in pay-per-click search advertising, the system must recommend advertisements to be displayed on search results pages. Each advertiser bids a fixed amount that is paid only when the user clicks on their ad. If we wish to optimize the expected system profit, both the bids and the probability that the user will click on each ad must be taken into account. This probability depends on the relevance of each ad to the user and the placement of the different ads on the page. Since the different ads displayed compete for the user’s attention, the utility function depends on the entire set of ads displayed, and is not additive over the set (Gunawardana and Meek, 2008).

In all of these cases, it may be suboptimal to suggest items based solely on their predicted rating. While it is certainly beneficial to recommend relevant items, other considerations are also important. For example, in the e-commerce scenario, given two items that the system perceives as equally relevant, suggesting the item with the higher profit can further increase revenue. In the online news agency case, recommending longer stories may be beneficial, because reading them will keep the user in the website longer. In the subscription service, recommending items that are harder for the user to reach without the recommender system may be beneficial.

Another common practice of recommendation systems is to suggest recommendations that provide the most “value” to the user. For example, recommending popular items can be redundant, as the user is probably already familiar with them. A recommendation of a preferred, yet unknown item can provide a much higher value for the user.

Such approaches can be viewed as instances of providing recommendations that maximize some utility function that assigns a value to each recommendation. Defining the correct utility function for a given application can be difficult (Braziunas and Boutilier, 2005), and typically system designers make simplifying assumptions about the user utility function. In the e-commerce case the utility function is typically the profit resulting from recommending an item, and in the news scenario the utility can be the expected time for reading a news item, but these choices ignore the effect of the resulting recommendations on long-term profits. When we are interested in novel recommendations, the utility can be the log of the inverse popularity of an item, modeling the amount of new information in a recommended item (Shani et al., 2005), but this ignores other aspects of user-utility such as the diversity of recommendations.

In fact, it is possible to view many recommendation tasks, such as providing novel or serendipitous recommendations as maximizing some utility function. Also, the “recommend good items” of the previous section can be considered as optimizing for a utility function assigning a value of 1 to each successful recommendation. In this paper, due to the popularity of the former task, we choose to keep the two tasks distinct.

### 3.3 Predicting Ratings

In some cases, a system is required to predict the user ratings over a given set of items. For example, in the Netflix website, when the user is browsing the list of new releases, the system assigns a predicted rating for each movie. In CNET,<sup>3</sup> a website offering electronic product reviews, users can

---

3. This can be found at [www.cnet.com](http://www.cnet.com).

search for, say, laptops that cost between \$400 and \$700. The system adds to some laptops in the list an automatically computed rating, based on the laptop features.

It is arguable whether this task is indeed a recommendation task. However, many researchers in the recommendations system community attempting to find good algorithms for this task. Examples include the Netflix competition, which was warmly embraced by the research community, and the numerous papers on predicting ratings on the Netflix or MovieLens<sup>4</sup> data sets.

While such systems do not provide lists of recommended items, predicting that the user will rate an item highly can be considered an act of recommendation. Furthermore, one can view a predicted high rating as a recommendation to use the item, and a predicted low rating as a recommendation to avoid the item. Indeed, it is common practice to use predicted ratings to generate a list of recommendations. Below, we will present several arguments of cases where this common practice may be undesirable.

## 4. Evaluation Protocols

We now discuss an experimental protocol for evaluating and choosing recommendation algorithms. We review several requirements to ensure that the results of the experiments are statistically sound. We also describe several common pitfalls in such experimental settings. This section reviews the evaluation protocols in related areas such as machine learning and information retrieval, highlighting practices relevant to evaluating recommendation systems. The reader is referred to publications in these fields for more detailed discussions (Salzberg, 1997; Demšar, 2006; Voorhees, 2002a).

We begin by discussing online experiments, which can measure the real performance of the system. We then argue that offline experiments are also crucial, because online experiments are costly in many cases. Therefore, the bulk of the section discusses the offline experimental setting in detail.

### 4.1 Online Evaluation

In the recommendation and utility optimization tasks, the designer of the system wishes to influence the behavior of users. We are therefore interested in measuring the change in user behavior when interacting with different recommendation systems. For example, if users of one system follow the recommendations more often (in the case of the “recommend good items” task), or if the utility gathered from users of one system exceeds utility gathered from users of the other system (in the utility optimization task), then we can conclude that one system is superior to the other, all else being equal. In the case of ratings prediction tasks, the goal is to provide information to support user browsing and search. Once again, the value of such predictions can depend on a variety of factors such as the user’s intent (e.g., how specific their information needs are, how much novelty vs. how much risk they are seeking), the user’s context (e.g., what items they are already familiar with, how much they trust the system), and the interface through which the predictions are presented.

For this reason, many real world systems employ an online testing system (Kohavi et al., 2009), where multiple algorithms can be compared. Typically, such systems redirect a small percentage of the traffic to each different recommendation engine, and record the users interactions with the different systems. There are a few considerations that must be made when running such tests. For example, it is important to sample (redirect) users randomly, so that the comparisons between

---

4. This can be found at [www.netflixprize.com](http://www.netflixprize.com).

alternatives are fair. It is also important to single out the different aspects of the recommenders. For example, if we care about algorithmic accuracy, it is important to keep the user interface fixed. On the other hand, if we wish to focus on a better user interface, it is best to keep the underlying algorithm fixed.

However, in a multitude of cases, such experiments are very costly, since creating online testing systems may require much effort. Furthermore, we would like to evaluate our algorithms before presenting their results to the users, in order to avoid a negative user experience for the test users. For example, a test system that provides irrelevant recommendations, may discourage the test users from using the real system ever again. Finally, designers that wish to add a recommendation system to their application before its deployment do not have an opportunity to run such tests.

For these reasons, it is important to be able to evaluate the performance of algorithms in an offline setting, assuming that the results of these offline tests correlate well with the online behavior of users.

## 4.2 Offline Experimental Setup

As described above, the goal of the offline evaluation is to filter algorithms so that only the most promising need undergo expensive online tests. Thus, the data used for the offline evaluation should match as closely as possible the data the designer expects the recommender system to face when deployed online. Care must be exercised to ensure that there is no bias in the distributions of users, items and ratings selected. For example, in cases where data from an existing system (perhaps a system without a recommender) is available, the experimenter may be tempted to pre-filter the data by excluding items or users with low counts, in order to reduce the costs of experimentation. In doing so, the experimenter should be mindful that this involves a trade-off, since this introduces a systematic bias in the data. If necessary, randomly sampling users and items may be a preferable method for reducing data, although this can also introduce other biases into the experiment (e.g., this could tend to favor algorithms that work better with more sparse data).

In order to evaluate algorithms offline, it is necessary to simulate the online process where the system makes predictions or recommendations, and the user corrects the predictions or uses the recommendations. This is usually done by recording historical user data, and then hiding some of these interactions in order to simulate the knowledge of how a user will rate an item, or which recommendations a user will act upon.

There are a number of ways to choose the ratings/selected items to be hidden. Once again, it is preferable that this choice be done in a manner that simulates the target application as closely as possible. We discuss these concerns explicitly for the case of selecting used items for hiding in the evaluation of recommendation tasks, and note that the same considerations apply when selecting ratings to hide for evaluation of ratings prediction tasks.

Our goal is to simulate sets of past user selections that are representative of what the system will face when deployed. Ideally, if we have access to time-stamps for user selections, we can randomly sample test users, randomly sample a time just prior to a user action, hide all selections (of all users) after that instant, and then attempt to recommend items to that user. This protocol requires changing the set of given information prior to each recommendation, which can be computationally quite expensive. A cheaper alternative is to sample a set of test users, then sample a single test time, and hide all items after the sampled test time for each test user. This simulates a situation where the recommender system is “trained” as of the test time, and then makes recommendations without

taking into account any new data that arrives after the test time. Another alternative is to sample a test time for each test user, and hide the test user's items after that time, without maintaining time consistency across users. This effectively assumes that it is the sequence in which items are selected, and not the absolute times when they are selected that is important. A final alternative is to ignore time; We sample a set of test users, then sample the number  $n_a$  of items to hide for each user  $a$ , then sample  $n_a$  items to hide. This assumes that the temporal aspects of user selections are unimportant. All three of the latter alternatives partition the data into a single training set and single test set. It is important to select an alternative that is most appropriate for the domain and task of interest, rather than the most convenient one.

A common protocol used in many research papers is to use a fixed number of known items or a fixed number of hidden items per test user (so called “given  $n$ ” or “all but  $n$ ” protocols). This protocol is useful for diagnosing algorithms and identifying in which cases they work best. However, when we wish to make decisions on the algorithm that we will use in our application, we must ask ourselves whether we are truly interested in presenting recommendations for users who have rated exactly  $n$  items, or are expected to rate exactly  $n$  items more. If that is not the case, then results computed using these protocol have biases that make them difficult to use in predicting the outcome of using the algorithms online.

The evaluation protocol we suggest above generates a test set (Duda and Hart, 1973) which is used to obtain held-out estimates for algorithm performance, using performance measures which we discuss below. Another popular alternative is to use cross-validation (Stone, 1974), where the data is divided into a number of partitions, and each partition in turn is used as a test set. The advantages of the cross-validation approach are to allow the use of more data in ranking algorithms, and to take into account the effect of training set variation. In the case of recommender systems, the held-out approach usually yields enough data to make reliable decisions. Furthermore, in real systems, the problem of variation in training data is avoided by evaluating systems trained on the historical data specific to the task at hand. In addition, there is a risk that since the results on the different data partitions are not independent of each other, pooling the results across partitions for ranking algorithms can lead to statistically unjustified decisions (Bengio and Grandvalet, 2004).

### 4.3 Making Reliable Choices

When choosing between algorithms, it is important that we can be confident that the algorithm that we choose will also be a good choice for the yet unseen data the system will be faced with in the future. As we explain above, we should exercise caution in choosing the data so that it would be most similar to the online application. Still, there is a possibility that the algorithm that performed best on this test set did so because the test set was fortuitously suitable for that algorithm. To reduce the possibility of such statistical mishaps, we must perform significance testing on the results.

Typically we compute a significance level or  $p$ -value—the probability that the obtained results were due to luck. Generally, we will reject the null hypothesis that algorithm  $A$  is no better than algorithm  $B$  if the  $p$ -value is above 0.05 (or below 95% confidence). That is, if the probability that the observed ranking is achieved by chance exceeds 0.05. More stringent significance levels (e.g., 0.01 or even lower) can be used in cases where the cost of making the wrong choice is higher.

In order to perform a significance test that algorithm  $A$  is indeed better than algorithm  $B$ , we require the results of several independent experiments comparing  $A$  and  $B$ . The protocol we have chosen in generating our test data ensures that we will have this set of results. Assuming that test

users are drawn independently from some population, the performance measures of the algorithms for each test user give us the independent comparisons we need. However, when recommendations or predictions of multiple items are made to the same user, it is unlikely that the resulting per-item performance metrics are independent. Therefore, it is better to compare algorithms on a per-user case. Approaches for use when users have not been sampled independently also exist, and attempt to directly model these dependencies (see, e.g., Larocque et al. 2007). Care should be exercised when using such methods, as it can be difficult to verify that the modeling assumptions that they depend on hold in practice.

Given such paired per-user performance measures for algorithms  $A$  and  $B$  the simplest test of significance is the sign test (Demšar, 2006). In this test, we count the number of users for whom algorithm  $A$  outperforms algorithm  $B$  ( $n_A$ ) and the number of users for whom algorithm  $B$  outperforms algorithm  $A$  ( $n_B$ ). The probability that  $A$  is not truly better than  $B$  is estimated as the probability of at least  $n_A$  out of  $n_A + n_B$  0.5-probability Binomial trials succeeding (that is,  $n_A$  out of  $n_A + n_B$  fair coin-flips coming up “heads”).

$$pr(\text{successes} \geq n_A | A = B) = 0.5^{n_A+n_B} \sum_{k=n_A}^{n_A+n_B} \frac{(n_A + n_B)!}{k!(n_A + n_B - k)!}.$$

The sign test is an attractive choice due to its simplicity, and lack of assumptions over the distribution of cases. Still this test may lead to mislabeling of significant results as insignificant when the number of test points is small. In these cases, the more sophisticated Wilcoxon signed rank test can be used (Demšar, 2006). As mentioned in Section 4.2, cross-validation can be used to increase the amount of data, and thus the significance of results, but in this case the results obtained on the cross-validated test sets are no longer independent, and care must be exercised to ensure that our decisions account for this (Bengio and Grandvalet, 2004). Also, model-based approaches (e.g., Goutte and Gaussier, 2005) may be useful when the amount of data is small, but once again, care must be exercised to ensure that the model assumptions are reasonable for the application at hand.

Another important consideration is the effect of evaluating multiple versions of algorithms. For example, an experimenter might try out several variants of a novel recommender algorithm and compare them to a baseline algorithm until they find one that passes a sign test at the  $p = 0.05$  level and therefore infer that their algorithm improves upon the baseline with 95% confidence. However, this is not a valid inference. Suppose the experimenter evaluated ten different variants all of which are statistically the same as the baseline. If the probability that any one of these trials passes the sign test mistakenly is  $p = 0.05$ , the probability that at least one of the ten trials passes the sign test mistakenly is  $1 - (1 - 0.05)^{10} = 0.40$ . This risk is colloquially known as “tuning to the test set” and can be avoided by separating the test set users into two groups—a development (or tuning) set, and an evaluation set. The choice of algorithm is done based on the development test, and the validity of the choice is measured by running a significance test on the evaluation set.

A similar concern exists when ranking a number of algorithms, but is more difficult to circumvent. Suppose the best of  $N + 1$  algorithms is chosen on the development test set. We can have a confidence  $1 - p$  that the chosen algorithm is indeed the best, if it outperforms the  $N$  other algorithms on the evaluation set with significance  $1 - (1 - p)^{1/N}$ . This is known as the Bonferroni correction, and should be used when pair-wise significant tests are used multiple times. Alternatively, the Friedman test for ranking can be used (Demšar, 2006).

## 5. Evaluating Tasks

An application designer that wishes to employ a recommendation system typically knows the purpose of the system, and can map it into one of the tasks defined above—recommendation, utility optimization, and ratings prediction. Given such a mapping, the designer must now decide which evaluation metric to use in order to rank a set of candidate recommendation algorithms. It is important that the metric match the task, to avoid an inappropriate ranking of the candidates.

Below we provide an overview of a large number of evaluation metrics that have been suggested in the recommendation systems literature. For each such metric we identify its important properties and explain why is it most appropriate for the given task. For each task we also explain a possible evaluation scenario that can be used to evaluate the various algorithms.

### 5.1 Predicting Ratings

In this task, the system must provide a set of predicted ratings, and is evaluated on the accuracy of these predictions. This is the most common scenario in the evaluation of regression and classification algorithms in the machine learning and statistics literature (Duda and Hart, 1973; Stone, 1974; Bengio and Grandvalet, 2004). Many evaluation metrics that originated in that literature have been applied here.

Most notably, the Root of the Mean Square Error (RMSE) is a popular method for scoring an algorithm. If  $p_{i,j}$  is the predicted rating for user  $i$  over item  $j$ , and  $v_{i,j}$  is the true rating, and  $K = \{(i, j)\}$  is the set of hidden user-item ratings then the RMSE is defined as:

$$\sqrt{\frac{\sum_{(i,j) \in K} (p_{i,j} - v_{i,j})^2}{n}}.$$

Other variants of this family are the Mean Square Error (which is equivalent to RMSE) and Mean Average Error (MAE), and Normalized Mean Average Error (NMAE) (Herlocker et al., 2004). RMSE tends to penalize larger errors more severely than the other metrics, while NMAE normalizes MAE by the range of the ratings for ease of comparing errors across domains.

RMSE is suitable for the prediction task, because it measures inaccuracies on all ratings, either negative or positive. However, it is most suitable for situations where we do not differentiate between errors. For example, in the Netflix rating prediction, it may not be as important to properly predict the difference between 1 and 2 stars as between 2 and 3 stars. If the system predicts 2 instead of the true 1 rating, it is unlikely that the user will perceive this as a recommendation. However, a predicted rating of 3 may seem like an encouragement to rent the movie, while a prediction of 2 is typically considered negative. It is arguable that the space of ratings is not truly uniform, and that it can be mapped to a uniform space to avoid such phenomena.

### 5.2 Recommending Good Items

For the task of recommending items, typically we are only interested in binary ratings, that is, either the item was selected (1) or not (0). Compared to ratings data sets, where users typically rate only a very small number of items, making the data set extremely sparse, binary selection data sets are dense, as each item was either selected or not by the user. An example of such data sets are news story click streams, where we set a value of 1 for each item that was visited, and a value of 0

|               | Recommended         | Not recommended     |
|---------------|---------------------|---------------------|
| Preferred     | True-Positive (tp)  | False-Negative (fn) |
| Not preferred | False-Positive (fp) | True-Negative (tn)  |

Table 1: Classification of the possible result of a recommendation of an item to a user.

elsewhere. The task is to provide, given an existing list of items that were viewed, a list of additional items that the user may want to visit.

As we have explained above, these scenarios are typically not symmetric. We are not equally interested in good and bad items; the task of the system is to suggest good items, not to discourage the use of bad items. We can classify the results of such recommendations using Table 1.

We can now count the number of examples that fall into each cell in the table and compute the following quantities:

$$\begin{aligned} \text{Precision} &= \frac{\#tp}{\#tp + \#fp}, \\ \text{Recall (True Positive Rate)} &= \frac{\#tp}{\#tp + \#fn}, \\ \text{False Positive Rate (1 - Specificity)} &= \frac{\#fp}{\#fp + \#tn}. \end{aligned}$$

Typically we can expect a trade off between these quantities—while allowing longer recommendation lists typically improves recall, it is also likely to reduce the precision. In some applications, where the number of recommendations that are presented to the user is not preordained, it is therefore preferable to evaluate algorithms over a range of recommendation list lengths, rather than using a fixed length. Thus, we can compute curves comparing precision to recall, or true positive rate to false positive rate. Curves of the former type are known simply as precision-recall curves, while those of the latter type are known as a Receiver Operating Characteristic<sup>5</sup> or ROC curves.

While both curves measure the proportion of preferred items that are actually recommended, precision-recall curves emphasize the proportion of recommended items that are preferred while ROC curves emphasize the proportion of items that are not preferred that end up being recommended.

We should select whether to use precision-recall or ROC based on the properties of the domain and the goal of the application; suppose, for example, that an online video rental service recommends DVDs to users. The precision measure describes what proportion of their recommendations were actually suitable for the user. Whether the unsuitable recommendations represent a small or large fraction of the unsuitable DVDs that could have been recommended (that is, the false positive rate) may not be as relevant.

On the other hand, consider a recommender system for an online dating site. Precision describes what proportion of the suggested pairings for a user result in matches. The false positive rate describes what proportion of unsuitable candidates are paired with the active user. Since presenting unsuitable candidates can be especially undesirable in this setting, the false positive rate could be the most important factor.

---

5. A reference to their origins in signal detection theory.

Given two algorithms, we can compute a pair of such curves, one for each algorithm. If one curve completely dominates the other curve, the decision about the winning algorithm is easy. However, when the curves intersect, the decision is less obvious, and will depend on the application in question. Knowledge of the application will dictate which region of the curve the decision will be based on. For example, in the “recommend some good items” task it is likely that we will prefer a system with a high precision, while in the “recommend all good items” task, a higher recall rate is more important than precision.

Measures that summarize the precision recall of ROC curve such as F-measure (Rijsbergen, 1979) and the area under the ROC curve (Bamber, 1975) are useful for comparing algorithms independently of application, but when selecting an algorithm for use in a particular task, it is preferable to make the choice based on a measure that reflects the specific needs at hand.

### 5.2.1 PRECISION-RECALL AND ROC FOR MULTIPLE USERS

When evaluating precision-recall or ROC curves for multiple test users, a number of strategies that can be employed in aggregating the results. The simplest is to aggregate the hidden ratings from the test set into a set of user-item pairs, generate a ranked list of user-item pairs by combining the recommendation lists for the test users, and then compute the precision-recall or ROC curve on this aggregated data.

This aggregation process assumes that we have a means of comparing recommendations made to different users in order to combine the recommendation lists into a single ranked list. Computing ROC curves in this manner treats the recommendations of different items to each user as being independent detection or classification tasks, and the resulting curve is termed a global ROC (GROC) curve (Schein et al., 2002).

A second approach is to compute the precision and recall (or true positive rate and false positive rate) at each recommendation list length  $N$  for each user, and then compute the average precision and recall (or true positive rate and false positive rate) at each  $N$  (Sarwar et al., 2000). The resulting curves are particularly valuable because they prescribe a value of  $N$  for each achievable precision and recall (or true positive rate and false positive rate), and conversely, can be used to estimate performance at a given  $N$ . Thus, this approach is useful in the “recommend some good items” scenario, where one important decision is the length of the recommendation list, by comparing performances along different candidate points along the curves. An ROC curve obtained in this manner is termed a Customer ROC (CROC) curve (Schein et al., 2002).

A third approach is to compute a precision-recall curve (or ROC curve) for each user and then average the resulting curves over users. This is the usual manner in which precision-recall curves are computed in the information retrieval community, and in particular in the influential TREC competitions (Voorhees, 2002b). This method is more relevant in the “recommend all good items” sub-task, if the system provides the user with all available recommendations and the user then scans the list linearly, marking each scanned item as relevant or not. The system can then compute the precision of the items scanned so far, and use the precision recall curve to give the user an estimate of what proportion of the good items have yet to be found.

## 5.3 Optimize Utility

Estimating the utility of a list of recommendations requires a model of the way users interact with the recommendations. For example, if a movie recommender system presents the DVD cover im-

ages of the top five recommendations prominently arranged horizontally across the top of the screen, the user will probably observe them all and select the items of interest. However, if all the recommendations are presented in a textual list several pages long, the user will probably scan down the list and abandon their scan at some point. In the first case, utility delivered by the top five recommendations actually selected would be a good estimate of expected utility, while in the second case, we would have to model the way users scan lists.

The half-life utility score of Breese et al. (1998) suggested such a model. It postulates that the probability that the user will select a relevant item drops exponentially down the list.

This approach evaluates an unbounded recommendation list, that potentially contains all the items in the catalog. Given such a list we assume that the user looks at items starting from the top. We then assume that an item at position  $k$  has a probability of  $\frac{1}{2^{(k-1)/(\alpha-1)}}$  of being viewed, where  $\alpha$  is a half life parameter, specifying the location of the item in the list with 0.5 probability of being viewed.

In the binary case of the recommendation task the half-life utility score is computed by:

$$\begin{aligned} R_a &= \sum_j \frac{1}{2^{(idx(j)-1)/(\alpha-1)}}, \\ R &= \frac{\sum_a R_a}{\sum_a R_a^{max}}, \end{aligned}$$

where the summation in the first equation is over the preferred items only,  $idx(j)$  is the index of item  $j$  in the recommendation list, and  $R_a^{max}$  is the score of the best possible list of recommendations for user  $a$ .

More generally, we can plug any utility function  $u(a, j)$  that assigns a value to a user item pair into the half-life utility score, obtaining the following formula:

$$R_a = \sum_j \frac{u(a, j)}{2^{(idx(j)-1)/(\alpha-1)}}.$$

Now,  $R_a^{max}$  is the score for the list of the recommendation where all the observed items are ordered by decreasing utility. In applications where the probability that a user will select the  $idx$ th item if it is relevant is known, a further generalization would be to use these known probabilities instead of the exponential decay.

#### 5.4 Fixed Recommendations Lists

When users add movies to their queues in Netflix, the system presents a list of 10 movies that they may like. However, when users choose to see recommendations (by clicking “movies that you will love”) the system presents all the possible recommendations. If there are too many recommended movies to fit a single page, the system allows the user to move to the next page of recommendations.

These two different usage scenarios illustrate a fundamental difference between recommendation applications—in the first, the system is allowed to show a small, fixed number of recommendations. In the second, the system provides as many recommendations as it can. Even though the two cases match a single task—the “recommend good items” task—there are several important distinctions that arise. It is important to evaluate the two cases properly.

When the system is required to present a list with a small, fixed size, that is known *a priori*, methods that present curves (precision-recall), or methods that evaluate the entire list (half-life

utility score), become less appropriate. For example, a system may get a relatively high half-life utility score, only due to items that fall outside the fixed list, while another system that selects all the items in the list correctly, and uninteresting items elsewhere, might get a lower score. Precision-recall curves are typically used to help us select the proper list length, where the precision and recall reach desirable values.

Another important difference, is that for a small list, the order of items in the list is less important, as we can assume that the user looks at all the items in the list. Moreover, many of these lists are presented in a horizontal direction, which also reduces the importance of properly ordering the items.

In these cases, therefore, a more appropriate way to evaluate the recommendation system should focus on the first  $N$  movies only. In the “recommend good items” task this can be done, for example, by measuring the precision at  $N$ —the number of items that are interesting out of the recommended  $N$  items. In the “optimize utility” task, we can do so by measuring the aggregated utility (e.g., sum of utility) of the items that are indeed interesting within the  $N$  recommendations.

A final case is when we have unlimited recommendation lists in the “recommend good items” scenario, and we wish to evaluate the entire list. In this case, one can use the half-life utility score with a binary utility of 1 when the (hidden) item was indeed selected by the user, and 0 otherwise. In that case, the half-life utility score prefers a recommender system that places interesting items closer to the head of the list, but provides an evaluation for the entire list in a single score.

## 6. Empirical Evaluation

In some cases, two metrics may provide a different ranking of two algorithms. When one metric is more appropriate for the task at hand, using the other metric may result in selecting the wrong algorithm. Therefore, it is important to choose the appropriate evaluation metric for the task at hand.

In this section we provide some empirical examples of the phenomenon we describe above, that is, where different metrics rank algorithms differently. Below, we present examples where algorithms are ranked differently by two metrics, one of which is more appropriate for the task of interest.

### 6.1 Data Sets

We selected publicly available data sets which were naturally suited to the different recommendation tasks we have described above. We begin by describing the properties of each data set we used.

#### 6.1.1 PREDICTION TASK

For the prediction task we selected two data sets that contained ratings over items—the Netflix data set and the BookCrossing data set. In both cases, the prediction task is quite natural. Users of both systems may want to browse the collection of movies or books, and we would want to offer these users an estimated rating for the presented items.

**Netflix:** In 2004, the online movie rental company Netflix<sup>6</sup> announced a competition for improving its recommendation system. For the purpose of the competition, Netflix has released a data set containing 480,000 users ratings over 17,700 movies. Ratings are between 1 and 5 stars for each movie. The data set is very sparse—users mostly rated a small fraction of the available movies. In

---

6. This can be found at [www.netflix.com](http://www.netflix.com).

our experiments, as we are working with simple algorithms, we have reduced the data set to users who rated more than 100 movies, leaving us with 21,179 users, 17,415 movies, and 117 ratings per user on average. Thus, our results are not comparable to results published in the online competition scoreboard.

**BookCrossing:** The BookCrossing website<sup>7</sup> allows a community of book readers to share their interests in books, and to review and discuss books. Within that system users can provide ratings on the scale of 1 to 10 stars. The specific data set that we used was collected by a 4 week crawl during August and September 2004 (Ziegler et al., 2005). The data set contains 105,283 users and 340,556 books (we used just the subset containing explicit ratings). Average ratings for a user is 10. This data set is even more sparse than the Netflix data set that we used, as there are more items and less ratings per user.

Both data sets share some common properties. First, people watch many movies and read many books, compared with other domains. For example, most people experience with only a handful of laptop computers, and so cannot form an opinion on most laptops. Ratings are also skewed towards positive ratings in both cases, as people are likely to watch movies that they think they will like, and even more so in the case of books, which require a heavier investment of time.

There are also some distinctions between the data sets. Some people feel compelled to share their opinion about books and movies, without asking for a compensation. However, in the Netflix domain, providing ratings makes it easier to navigate the system and rent movies. Therefore, all users of Netflix have an incentive for providing ratings, while only people who like to share their views of books use the BookCrossing system. We can therefore expect that the ratings of the BookCrossing are less representative of the general population of book readers, than the ratings of Netflix user from the general population of DVD renters.

#### 6.1.2 RECOMMENDATION TASK

One instance of the “recommend good items” task is the case where, given a set of items that the user has used (bought, viewed), we wish to recommend a set of items that are likely to be used. Typically, data sets of usage are binary—an item was either used or wasn’t used by the user, and the data set is not sparse, because every item is either used or not used by every user. We used here a data set of purchases from supermarket retailer, and a stream of articles that were viewed in a news website.

**Belgian retailer:** This data set was collected from an anonymous Belgian retail supermarket store, collected over approximately 5 months, in three non-consecutive periods during 1999 and 2000. The data set is divided into baskets, and we cannot detect return users. There are 88,162 baskets, 16,470 distinct items, and 10 items in an average basket. We do not have access for item prices or profits, so we cannot optimize the retailer revenue. Therefore the task is to recommend more items that the user may want to add to the basket.

**News click stream:** This is a log of click-stream data of an Hungarian online news portal (Bodon, 2003). The data contains 990,002 sessions, 41,270 news stories, and an average of 8 stories for session. The task is, given the news items that a user has read so far, recommend more news items that the user will likely read.

---

7. This can be found at [www.bookcrossing.com](http://www.bookcrossing.com).

### 6.1.3 OPTIMIZING UTILITY TASK

**Ta-Feng supermarket:** A natural example for an application where optimizing utility is important is maximizing the revenues of a retail company. Such companies may provide recommendation for items, hoping that customers following these recommendations will produce higher revenue. In this case, a natural utility function is the revenue (or profit) from the purchase of an item. The Ta-Feng data set (Hsu et al., 2004) contains transaction information collected over 4 months from November, 2000 to February, 2001. There are 32,266 users and 23,812 items, where the average number of items bought by a user is 23. In this task, the utility function is the accumulated profit from selling an item—taking into account both the quantity and the profit per item.

## 6.2 Recommendation Algorithms

As the focus of this survey is on the correct evaluation of recommender systems, and not on sophisticated algorithms for computing recommendation lists, we limit ourselves to a set of very simple collaborative filtering algorithms. We do this because collaborative filtering is by far the most popular recommendation approach, and because we do not believe that it is appropriate to select the evaluation metric based on the recommendation approach (e.g., collaborative filtering vs. content based).

Moreover, we carefully selected algorithms that are better suited for different tasks, so that we could demonstrate that inappropriate choice of evaluation metric can lead to a bad choice of algorithm. As the algorithms that we choose are computationally intensive, we reduced the size of the data set in some cases, in order to reduce the computation time. This should not be done if it was important to realistically simulate the online case. Below, we present the different algorithms and our prior assumptions about their properties.

### 6.2.1 PEARSON CORRELATION

Typically, the input for a prediction task is a data set consisting of the ratings provided by  $n$  users for  $m$  items, where  $v_{i,j}$  is the rating of user  $i$  for item  $j$ . Given such a data set, the simplest collaborative filtering method computes the similarity of the active user  $a$  to all other users  $i$  in the data set, resulting in a score  $w(a,i)$ . Then, the predicted rating  $p_{a,j}$  for  $a$  over item  $j$  can be computed by:

$$p_{a,j} = \bar{v}_a + \kappa \sum_{i=1}^n w(a,i)(v_{i,j} - \bar{v}_i). \quad (1)$$

Perhaps the most popular method for computing the weights  $w(a,i)$  is by using the Pearson correlation coefficient (Resnick and Varian, 1997):

$$w(a,i) = \frac{\sum_j (v_{a,j} - \bar{v}_a)(v_{i,j} - \bar{v}_i)}{\sqrt{\sum_j (v_{a,j} - \bar{v}_a)^2 \sum_j (v_{i,j} - \bar{v}_i)^2}}$$

where the summations are only over the items that both  $a$  and  $i$  have rated. To reduce the computational overhead, we use in Equation 1 a neighborhood of size  $N$ .

This method is specifically designed for the prediction task, as it computes only a predicted score for each item of interest. However, in many cases people used this method for the recommendation task. This is typically done by predicting the scores for all possible items, and then ordering the items by decreasing predicted scores.

This popular usage may not be appropriate. For example, in the movie domain people may associate ratings with quality, as opposed to enjoyment, which is dependent on external factors such as mood, time of day, and so forth. As such, 5 stars movies may be complicated, requiring a substantial effort from the viewer. Thus, a user may rent many light effortless romantic comedies, which may only get a score of 3 stars, and only a few 5 star movies. While it is difficult to measure this effect without owning a rental store, we computed the average number of ratings for movies with different average rating (Figure 6.2.1). This figure may suggest that movies with higher ratings are not always watched more often than movies with lower ratings. If our assumption is true, a system that recommends items to add to the rental queue by order of decreasing predicted rating, may not do as well as a system that predicts the probability of adding a movie to the queue directly.

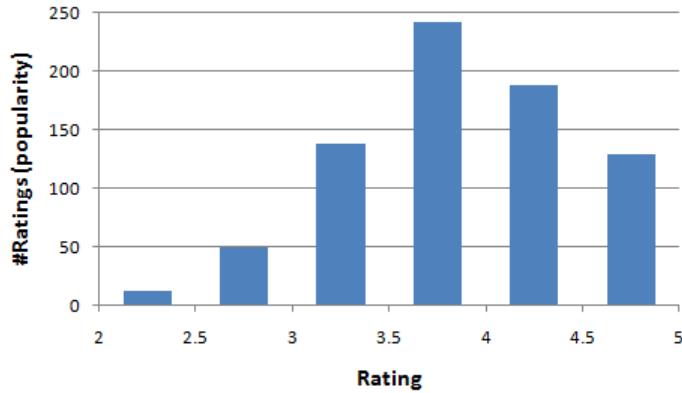


Figure 1: Computing the average number of ratings (popularity) of movies binned given their average ratings.

### 6.2.2 COSINE SIMILARITY

A second popular collaborative filtering method is the vector similarity metric (Salton, 1971) that measures the cosine angle formed by the two ratings vectors:

$$w(a, i) = \sum_{j \in I_{a,i}} \frac{v_{a,j}}{\sqrt{\sum_{k \in I_a} v_{a,k}^2}} \frac{v_{i,j}}{\sqrt{\sum_{k \in I_i} v_{i,k}^2}}.$$

When computing the cosine similarity, only positive ratings have a role, and negative ratings are discarded. Thus,  $I_i$  is the set of items that user  $i$  has rated positively and  $I_{a,i}$  is the set of items that both users rated positively. Also, the predicted score for a user is computed by:

$$p_{a,j} = \kappa \sum_{i=1}^n w(a, i) v_{i,j}.$$

In the case of binary data sets, such as the usage data sets that we selected for the recommendation task, the vector similarity method becomes:

$$w(a, i) = \frac{|I_{a,i}|}{\sqrt{|I_a|} \cdot \sqrt{|I_i|}}$$

where  $I_a$  is the set of items that  $a$  used, and  $I_{a,i}$  is the set of items that both  $a$  and  $i$  used. The resulting aggregated score can be considered as a non-calibrated measurement of the conditional probability  $pr(j|a)$ —the probability that user  $a$  will choose item  $j$ .

In binary usage data sets, the Pearson correlation method would compute similarity using all the items, as each item always has a rating. Therefore, the system would use all the negative “did not use” scores, which typically greatly outnumber the “used” scores. We can therefore expect that Pearson correlation in these cases will result in lower accuracy.

### 6.2.3 ITEM TO ITEM

The above two methods focused on computing a similarity between users, but another possible collaborative filtering alternative is to focus on the similarity between items. The simplest method for doing so is to use the maximum likelihood estimate for the conditional probabilities of items. Specifically, for the binary usage case, this translates to:

$$pr(j_1|j_2) = \frac{|J_{j_1,j_2}|}{|J_{j_2}|}$$

where  $J_j$  is the number of users who used item  $j$ , and  $J_{j_1,j_2}$  is the number of users that used both  $j_1$  and  $j_2$ . While this seems like a very simple estimation, similar estimations are successfully used in deployed commercial applications (Linden et al., 2003).

Typically, an algorithm is given as an input a set of items, and needs to produce a list of recommendations. In that case, we can compute for the conditional probability of each target item given each observed item, and then aggregate the results over the set of given items. In many cases, choosing the maximal estimate has given the best results (Kadie et al., 2002), so we aggregate estimations using a max operator in our experiments.

### 6.2.4 EXPECTED UTILITY

As optimizing utilities is by far the least explored recommendation task, we choose here to propose a new algorithm that is designed specifically for this task. Intuitively, if the task requires lists that optimize a utility function  $u(a, j)$ , an obvious method is to order the recommendation by decreasing expected utility:

$$E[j|a] = \tilde{pr}(j|a) \cdot \tilde{u}(a, j)$$

where  $\tilde{pr}$  is a conditional probability estimate and  $\tilde{u}$  is a utility estimate.

One way to compute the two estimates is by using two different algorithms—a recommendation algorithm for estimating  $\tilde{pr}$  and a prediction algorithm for estimating  $\tilde{u}$ , and then combining their output.

## 6.3 Experimental Results

Below, we present several examples where different evaluation metrics rank two algorithms differently. We argue that in these cases, using an improper evaluation metric will lead to the selection of an inferior algorithm.

|         | Netflix | BookCrossing |
|---------|---------|--------------|
| Pearson | 1.07    | 3.58         |
| Cosine  | 1.90    | 4.5          |

Table 2: RMSE scores for Pearson correlation and Cosine similarity on the Netflix domain (ratings from 1 to 5) and the BookCrossing domain (ratings from 1 to 10).

### 6.3.1 PREDICTION VS. RECOMMENDATION

We begin by comparing Pearson correlation and Cosine similarity collaborative filtering algorithms over two tasks—the prediction task and the “recommend good items” task. Both algorithms used neighborhoods consisting of the closest 25 users.

First we evaluated the algorithms in predicting ratings on the Netflix and BookCrossing data sets, where we sampled 2000 test users and a randomly chosen number of test items per test user on each data set. Given Table 2, the algorithm of choice is clear—on both data sets, the predictions given by the Pearson correlation algorithm have lower RMSE scores, and the differences pass a sign test with  $p < 0.0001$ .

We then evaluated the two algorithms on the recommendation task on the Belgian retailer and news click stream data sets, where we again sampled 2000 test users and a randomly chosen number of test items per test user on each data set. Let us now evaluate the two algorithms on recommendation tasks. To do that, we computed precision-recall curves for the two algorithms on the Belgian retailer data set and the news click stream data set. This was done by computing precision and recall at 1, 3, 5, 10, 25, and 50 recommendations, and averaging the precisions and recalls at each number of recommendations. As Figure 2 shows, in both cases the recommendation lists generated by the Cosine similarity dominate the recommendation lists generated by the Pearson correlation algorithm in terms of precision. In the Belgian retailer data, Cosine similarity also has better recall than Pearson correlation across the board. On the news click stream data, Cosine similarity has better recall than Pearson correlation for 1, 3, and 5, recommendations. All these comparisons were significant according to a sign test with  $p < 0.0001$ . Therefore, in these cases, one would select the Cosine algorithm as the most appropriate choice.

This experiment shows that an algorithm that is uniformly better at predicting ratings on ratings data sets is not necessarily better at making recommendations on usage data sets. This suggests that it is possible that an algorithm that is better at predicting ratings could be worse at predicting usage in the same domain as well. An interesting experiment would be, given both ratings and usage data over the same users and items, to see whether algorithms that generate recommendation lists by decreasing order of predicted ratings do as well in the recommendation task over the usage data. Unfortunately, we are unaware of any public data set that contains both types of information. Nevertheless, companies such as Amazon or Netflix collect data both of user purchases and of user ratings over items. These companies can therefore make the appropriate decision for the recommendation task at hand.

It is also possible that websites that support multiple recommendation tasks should use different algorithms for the different tasks. For example, the Netflix website contains a prediction task (e.g., for new releases), and two recommendation tasks—a fixed list of recommendations when adding items to the rental queue, and an unlimited list of recommendations in the “movie you will like”

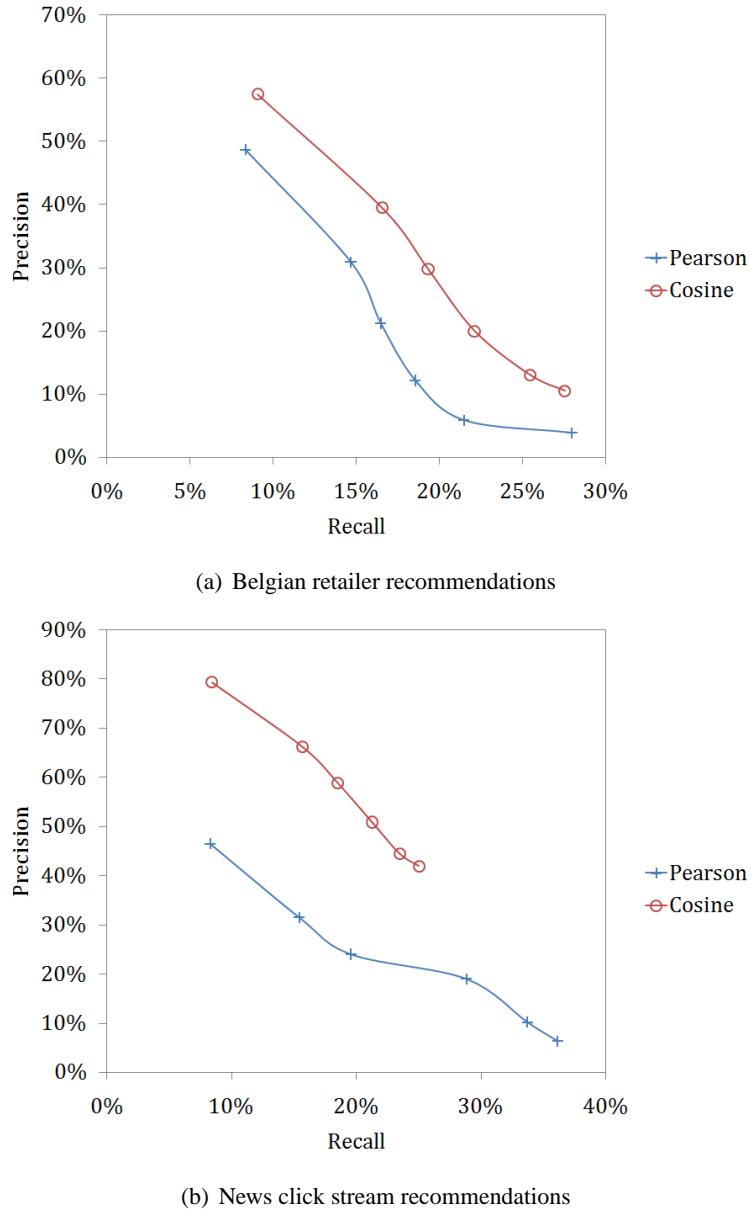


Figure 2: Comparing recommendations generated by Pearson correlation and Cosine similarity. In both cases, the recommendation list is ordered by decreasing predicted score.

section. It may well be that different algorithms that were trained over different data sets (ratings vs. rentals) may rank differently in different tasks. Deciding on the best recommendation engine based solely on RMSE in the prediction task may lead to worse recommendation lists in the two other cases.

### 6.3.2 RECOMMENDATION VS. UTILITY MAXIMIZATION

In many retail applications, where the retailer is interested in maximizing profits, people may still train their algorithm on usage data solely, and evaluate them using recommendation oriented metrics, such as precision-recall. For example, even though a retailer website wishes to maximize its profit, it may use a binary data set of items that were bought by users to generate recommendations of the type “people who bought this item also bought ...”.

To evaluate the performance of such an approach, we used an item-item recommendation system to generate recommendations for the Ta-Feng data set. Alternatively, one can order the items by expected utility. To compute an estimate of expected utility, we interpreted the normalized predicted score of each recommended item as the probability that the user would actually buy that item. In the case where the recommender predicts numerical ratings, we normalize by the highest possible rating and treat the result as a probability distribution. For example, if the highest rating is 5 and the algorithm predicts a score of 4.5 for a specific user we assume the probability that the user will use the item is 0.9. We then use the average profit earned from each item to predict the profit that would be obtained from each item if the active user bought it. Multiplying the probability that the user would buy an item by the profit that would result if the user bought the item yielded the required estimate of expected utility.

We then evaluated the two algorithms by comparing their precision-recall curves, which are shown in Figure 3. The curves were generated by evaluating precision and recall at 1, 3, 5, 10, 25, and 50 recommendations, and averaging the precisions and recalls at each number of recommendations. The averages were computed over 2000 users, and the item-item recommender outperformed the expected profit recommender in terms of both precision and recall at all points with  $p < 0.0001$ .

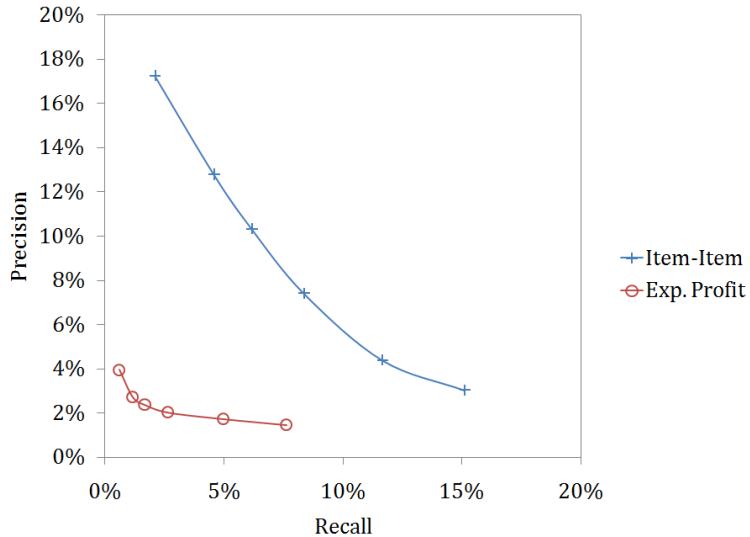


Figure 3: Comparing recommendations generated by the item-item recommender and the expected profit recommender on the Ta-Feng data set.

We then measured an half-life utility score where the utility of a correct recommendation was the profit from selling the correctly recommended item to the user. The results are shown in Table 3.

|             | Score |
|-------------|-------|
| Item-Item   | 0.01  |
| Exp. Profit | 0.05  |

Table 3: Comparing item-item vs. expected utility recommendations on the Ta-Feng data set with the half-life utility score. The utility of a correct recommendation was the profit from selling that item to that user, while the half-life was 5. The trends were similar for other choices of the half-life parameter.

Choosing a recommender based on classification performance would have resulted in an half-life utility score (expected profit) that was 20% of what could have been achieved with the correct choice. This difference is statistically significant with  $p < 0.001$ .

These results are, of course, not surprising—using a recommender that explicitly attempts to optimize expected profit gives a better expected profit measure. However, occasionally people that are interested in maximizing profit, providing maximum value to the users, or minimizing user effort, use precision-recall to choose a recommendation algorithm.

## 7. Discussion

Above, we discussed the major considerations that one should make when deciding on the proper evaluation metric for a given task. We now add some discussion, illustrating other conclusions that can be derived, and illuminating some other relevant topics.

### 7.1 Evaluating Complete Recommender Systems

This survey focuses on the evaluation of recommendation algorithms. However, the success of a recommendation system does not depend solely on the quality of the recommendation algorithm. Such systems typically attempt to modify user behavior which is influenced by many other parameters, most notably, by the user interface. The success of the deployed system in influencing users can be measured through the change in user behavior, such as the number of recommendations that are followed, or the change in revenue.

Decisions about the interface by which users view recommendations are critical to the success of the system. For example, recommendations can be located in different places in the page, can be displayed horizontally or vertically, can be presented through images or text, and so forth. These decisions can make a significant impact, no smaller than the quality of the underlying algorithm, on the success of a system.

When the application is centered around the recommendation system, it is important to select the user interface together with the recommendation algorithm. In other cases, the recommendation system is only a supporting system for the application. For example, an e-commerce website is centered around the item purchases, and a news website is centered around the delivery of news stories. In both cases, a recommender system may be employed to help the users navigate, or to increase sales. It is likely that the recommendations are not the major method for browsing the collection of items.

When the recommender system is only a supporting system, the designer of the application will probably make the decision about the user interface without focusing on positioning recommendations where they have the most influence. In such cases, which we believe to be very common, the developer of the recommender system is constrained by the pre-designed interface, and in many cases can therefore only decide on the best recommendation algorithm, and in some cases perhaps the length of the recommendation list. This paper is targeted at researchers and developers who are making decisions about algorithms, not about the user interface. Designing a good user interface is an interesting and challenging problem, but it is outside the scope of this survey (see, e.g., Pu and Chen 2006).

## 7.2 Eliciting Utility Functions

A simple way to avoid the need to classify recommendation algorithms, is to assume that we are always optimizing some utility function. This utility function can be the user utility for items (see, e.g., Kumar et al. 1998, Price and Messinger 2005 and Hu and Pu 2009), or it can be the application utility. We can then ask users about their true utility function, or design a utility function that captures the goal of the application, and always choose the algorithm that maximizes this utility.

However, such a view is misleading; eliciting user utilities can be a very difficult task (see, e.g., Braziunas and Boutilier 2005, citealtChajewska and Huang 2008). For example, the value that the user is willing to invest in a laptop depends on a multitude of elements, such as her income, her technical knowledge, the intended use of the laptop and so forth. Indeed, eliciting such functions is the focus of active research, for example by presenting users with forced choices. Therefore, expecting that we will have access to a good estimate of the user utility function is unrealistic—estimating this function may be no easier than coming up with good recommendations.

Furthermore, even when the application designer understands the application utility function, gathering utilities from users may be very difficult. For example, in the Netflix domain, the business model may be to keep the users subscribed. Therefore, the utility of a movie for a user (from the website perspective) is not whether the user enjoys the movie, but rather whether the user will maintain her subscription if the movie is suggested to her. Clearly, most users will not want to answer such questions, and many may not know the answer themselves.

For this reason, when the utility function is unclear, the best we can do is to maximize the number of useful items that we suggest to the user. As such, the recommendation task cannot be viewed as a sub-task for utility optimization.

## 7.3 Implicit vs. Explicit ratings

Our classification of recommendation tasks sheds some light over another, commonly discussed subject in recommender systems, namely, implicit ratings (Claypool et al., 2001; Oard and Kim, 1998). In many applications, people refer to data that was automatically collected, such as logs of web browsing, or records of product purchases, as an implicit indication for positive opinions over the items that were visited or purchased.

However, this perspective is appropriate only if the task is the prediction task, where we would like to know whether the user will have a positive or negative opinion over certain items. If the task is to recommend more items that the user may buy, given the items that the user has already bought, purchase data becomes an explicit indication. In this case, using ratings that users provide over items is an implicit indication to whether the user will buy the item.

For example, a user may have a positive opinion over many laptop computers, and may rate many laptops highly. However, most people buy only one laptop. In that case, recommending more laptops, based on the co-occurring high ratings, will be inappropriate. However, if we predict the probability of buying a laptop given that another laptop has already been bought, we can expect this probability to be low, and the other laptop will not be recommended.

## 8. Related Work

In the past, different researchers discussed various topics relevant to the evaluation of recommender systems.

Breese et al. (1998) were probably the first to provide a sound evaluation of a number of recommendation approaches over a collection of different data sets, setting the general framework of evaluating algorithms on more than a single real world data set, and a comparison of several algorithms in identical experiments. The practices that were illustrated in that paper are used in many modern publications.

Herlocker et al. (2004) provide an extensive survey of possible metrics for evaluation. They then compare a set of metrics, concluding that for some pairs of metrics, using both together will give very little additional information compared to using just one.

Another interesting contribution of that paper is a classification of recommendation engines from the user task perspective, namely, what are the reasons and motivations that a user has when interacting with a recommender system. As they are interested in user tasks, and we are interested in the system tasks, our classification is different, yet we share some similar tasks, such as the “recommend some good items” and “recommend all good items” tasks.

Finally, their survey attempted to cover as many evaluation metrics and user task variations as possible, we focus here on the appropriate metrics for the most popular recommendation tasks only.

Mcnee et al. (2003) explain why accuracy metrics alone are insufficient for selecting the correct recommendation algorithm. For example, users may be interested in the serendipity of the recommended items. One way to model serendipity is through a utility function that assigns higher values to “unexpected” suggestions. They also discuss the “usefulness” of recommendations. Many utility functions, such as the inverse log of popularity (Shani et al., 2005) attempt to capture this “usefulness”.

Ziegler et al. (2005) focus on another aspect of evaluation—considering the entire list together. This would allow us to consider aspects of a set of recommendations, such as diversification between items in the same list. Our suggested metrics consider only single items, and thus could not be used to evaluate entire lists. It would be interesting to see more evaluation metrics that provide observations over complete lists.

Celma and Herrera (2008) suggest looking at topological properties of the recommendation graph—the graph that connects recommended items. They explain how by looking at the recommendation graph one may understand properties such as the novelty of recommendations. It is still unclear how these properties correlate with the true goal of the recommender system, may it be to optimize revenue or to recommend useful items.

McLaughlin and Herlocker (2004) argue, as we do, that MAE is not appropriate for evaluating recommendation tasks, and that ratings are not necessarily indicative of whether a user is likely to watch a movie. The last claim can be explained by the way we view implicit and explicit ratings.

Some researchers have suggested taking a more holistic approach, and considering the recommendation algorithm within the complete recommendation system. For example, del Olmo and Gaudioso (2008), suggest that systems be evaluated only after deployment, through counting the number of successful recommendations. As we argue above, even in these cases, one is likely to evaluate algorithms offline, to avoid presenting recommendations that have poor quality for users, thus losing their trust.

## 9. Conclusion

In this paper we discussed how recommendation algorithms should be evaluated in order to select the best algorithm for a specific task from a set of candidates. This is an important step in the research attempt to find better algorithms, as well as in application design where a designer chooses an existing algorithm for their application. As such, many evaluation metrics have been used for algorithm selection in the past.

We review three core tasks of recommendation systems—the prediction task, the recommendation task, and the utility maximization task. Most evaluation metrics are naturally appropriate for one task, but not for the others. We discuss for each task a set of metrics that are most appropriate for selecting the best of the candidate algorithms.

We empirically demonstrate that in some cases two algorithms can be ranked differently by two metrics over the same data set, emphasizing the importance of choosing the appropriate metric for the task, so as not to choose an inferior algorithm.

We also describe the concerns that need to be addressed when designing offline and online experiments. We outline a few important measurements that one must take in addition to the score that the metric provides, as well as other considerations that should be taken into account when designing experiments for recommendation algorithms.

## References

- D. Bamber. The area above the ordinal dominance graph and the area below the receiver operating characteristic graph. *Journal of Mathematical Psychology*, 12:387–415, 1975.
- Y. Bengio and Y. Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of Machine Learning Research*, 5, 2004.
- F. Bodon. A fast APRIORI implementation. In *The IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, 2003.
- D. Braziunas and C. Boutilier. Local utility elicitation in GAI models. In *Proceedings of the Twenty-first Conference on Uncertainty in Artificial Intelligence*, pages 42–49, Edinburgh, 2005.
- J. S. Breese, D. Heckerman, and C. M. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI: Uncertainty in Artificial Intelligence*, pages 43–52, 1998.
- Ó. Celma and P. Herrera. A new approach to evaluating novel recommendations. In *RecSys '08: Proceedings of the 2008 ACM Conference on Recommender Systems*, 2008.
- M. Claypool, P. Le, M. Waseda, and D. Brown. Implicit interest indicators. In *Intelligent User Interfaces*, pages 33–40. ACM Press, 2001.

- F. Hernández del Olmo and E. Gaudioso. Evaluation of recommender systems: A new approach. *Expert Systems Applications*, 35(3), 2008.
- J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7, 2006.
- R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.
- C. Goutte and E. Gaussier. A probabilistic interpretation of precision, recall, and F-score, with implication for evaluation. In *ECIR '05: Proceedings of the 27th European Conference on Information Retrieval*, pages 345–359, 2005.
- A. Gunawardana and C. Meek. Aggregators and contextual effects in search ad markets. In *WWW Workshop on Targeting and Ranking for Online Advertising*, 2008.
- J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, 22(1), 2004.
- C. N. Hsu, H. H. Chung, and H. S. Huang. Mining skewed and sparse transaction data for personalized shopping recommendation. *Machine Learning*, 57(1-2), 2004.
- R. Hu and P. Pu. A comparative user study on rating vs. personality quiz based preference elicitation methods. In *IUI '09: Proceedings of the 13th International Conference on Intelligent User Interfaces*, 2009.
- S. L. Huang. Comparision of utility-based recommendation methods. In *The Pacific Asia Conference on Information Systems*, 2008.
- C. Kadie, C. Meek, and D. Heckerman. CFW: A collaborative filtering system using posteriors over weights of evidence. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 242–250, San Francisco, CA, 2002. Morgan Kaufmann.
- R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne. Controlled experiments on the web: survey and practical guide. *Data Mining and Knowledge Discovery*, 18(1), 2009.
- J. A. Konstan, S. M. McNee, C. N. Ziegler, R. Torres, N. Kapoor, and J. Riedl. Lessons on applying automated recommender systems to information-seeking tasks. In *Proceedings of the Twenty-First National Conference on Artifical Intelligence (AAAI)*, 2006.
- R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Recommendation systems: A probabilistic analysis. In *FOCS '98: Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, 1998.
- D. Larocque, J. Nevalainen, and H. Oja. A weighted multivariate sign test for cluster-correlated data. *Biometrika*, 94:267–283, 2007.
- G. Linden, B. Smith, and J. York. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1), 2003.

- M. R. McLaughlin and J. L. Herlocker. A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In *SIGIR '04: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2004.
- S. McNee, S. K. Lam, C. Guetzlaff, J. A. Konstan, and J. Riedl. Confidence displays and training in recommender systems. In *Proceedings of the 9th IFIP TC13 International Conference on Human Computer Interaction INTERACT*, pages 176–183. IOS Press, 2003.
- S. M. McNee, J. Riedl, and J. K. Konstan. Making recommendations better: an analytic model for human-recommender interaction. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, 2006.
- M. Montaner, B. López, and J. L. De La Rosa. A taxonomy of recommender agents on the internet. *Artificial Intelligence Review*, 19(4), 2003.
- D. Oard and J. Kim. Implicit feedback for recommender systems. In *The AAAI Workshop on Recommender Systems*, pages 81–83, 1998.
- B. Price and P. Messinger. Optimal recommendation sets: Covering uncertainty over user preferences. In *National Conference on Artificial Intelligence (AAAI)*, pages 541–548. AAAI Press / The MIT Press, 2005.
- P. Pu and L. Chen. Trust building with explanation interfaces. In *IUI '06: Proceedings of the 11th International Conference on Intelligent User Interfaces*, 2006.
- P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3), 1997.
- C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 1979.
- G. Salton. *The SMART Retrieval System—Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971.
- S. L. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3), 1997.
- B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Analysis of recommendation algorithms for e-commerce. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, 2000.
- J. B. Schafer, J. Konstan, and J. Riedl. Recommender systems in e-commerce. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce*, 1999.
- A. I. Schein, A. Popescul, L. H. Ungar, and D. M. Pennock. Methods and metrics for cold-start recommendations. In *SIGIR '02: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2002.
- G. Shani, D. Heckerman, and R. I. Brafman. An mdp-based recommender system. *Journal of Machine Learning Research*, 6:1265–1295, 2005.
- M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society B*, 36(1):111–147, 1974.

- E. M. Voorhees. The philosophy of information retrieval evaluation. In *CLEF '01: Revised Papers from the Second Workshop of the Cross-Language Evaluation Forum on Evaluation of Cross-Language Information Retrieval Systems*, 2002a.
- E. M. Voorhees. Overview of trec 2002. In *The 11th Text Retrieval Conference (TREC 2002), NIST Special Publication 500-251*, pages 1–15, 2002b.
- C. N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW '05: Proceedings of the 14th International Conference on the World Wide Web*, 2005.

Hacking and Gonzo, a publication by Amir Salihefendic since 2000

[Search >](#)

# How Hacker News ranking algorithm works

In this post I'll try to explain how [Hacker News](#) ranking algorithm works and how you can reuse it in your own applications. It's a very simple ranking algorithm and works surprisingly well when you want to highlight hot or new stuff.



I'm Amir Salihefendic  
founder and CEO of [Doist](#)

## Digging into news.arc code

Hacker News is implemented in Arc, a Lisp dialect coded by [Paul Graham](#). Hacker News is open source and the code can be found at [arclanguage.org](#). Digging through the news.arc code you can find the ranking algorithm which looks like this:

```
; Votes divided by the age in hours to the gravityth power.
; Would be interesting to scale gravity in a slider.

(= gravity* 1.8 timebase* 120 front-threshold* 1
  nourl-factor* .4 lightweight-factor* .3 )

(def frontpage-rank (s (o scorefn realscore) (o gravity gravity*))
  (* (/ (let base (- (scorefn s) 1)
        (if (> base 0) (expt base .8) base))
        (expt (/ (+ (item-age s) timebase*) 60) gravity)))
     (if (no (in s!type 'story 'poll)) 1
       (blank s!url)               nourl-factor*
       (lightweight s)             (min lightweight-factor*
                                     (contro-factor s))
       (contro-factor s))))
```

[Stay in Touch](#)

[Twitter @amix3k](#)

[amix@amix.dk](#)

[Blog RSS feed](#)

[Current Focus](#)

[Doist](#)

[Todoist](#)

[Wedoist](#)

[github.com/amix](#)

[Labels](#)

[AJAX and comet](#)

[amix.dk related](#)

[Announcements](#)

[Benchmarks](#)

[Books](#)

[Code](#)

[Code improvement](#)

[Code rewrite](#)

[Database](#)

[Design](#)

[Education](#)

[Interesting](#)

[JavaScript](#)

In essence the ranking performed by Hacker News looks like this:

```
Score = (P-1) / (T+2)^G

where,
P = points of an item (and -1 is to negate submitters vote)
T = time since submission (in hours)
G = Gravity, defaults to 1.8 in news.arc
```

As you see the algorithm is rather trivial to implement. In the upcoming section we'll see how the algorithm behaves.

Life

node.js

Orangoo

Plurk

Political

Posters

Presentations

Productivity

Psychology

Python

Reading list

Security

Skeletononz

Stuff

Tips

Todoist

VIM Editor

Wedoist

Archive

2015

2014

2013

2012

2011

2010

2009

2008

2007

2006

2005

2004

## Effects of gravity (G) and time (T)

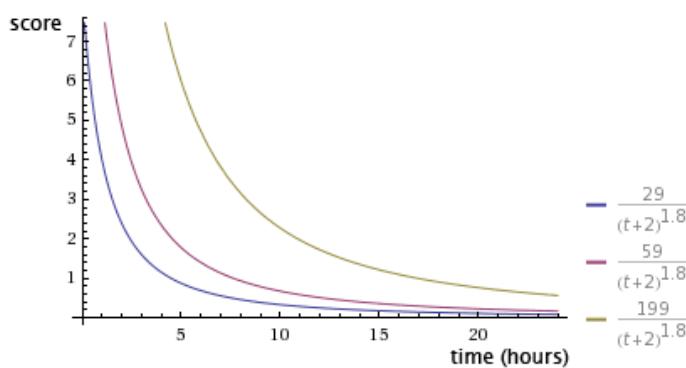
Gravity and time have a significant impact on the score of an item. Generally these things hold true:

- the score decreases as T increases, meaning that older items will get lower and lower scores

- the score decreases much faster for older items if gravity is increased

To see this visually we can plot the algorithm to [Wolfram Alpha](#).

How score is behaving over time

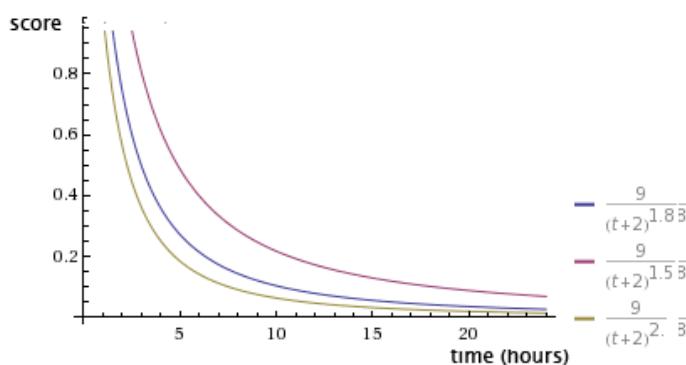


As you can see the score decreases a lot as time goes by, for example a 24 hour old item will have a very low score regardless of how many votes it got.

Plot query:

```
plot(
  (30 - 1) / (t + 2)^1.8,
  (60 - 1) / (t + 2)^1.8,
  (200 - 1) / (t + 2)^1.8
) where t=0..24
```

How gravity parameter behaves



As you can see by the graph the score decreases a lot faster the larger the gravity is.

Plotting query:

```
plot(
    (p - 1) / (t + 2)^1.8,
    (p - 1) / (t + 2)^0.5,
    (p - 1) / (t + 2)^2.0
) where t=0..24, p=10
```

## Python implementation

As already stated it's rather simple to implementing the score function:

```
def calculate_score(votes, item_hour_age, gravity=1.8):
    return (votes - 1) / pow((item_hour_age+2), gravity)
```

The most crucial aspect is understanding how the algorithm behaves and how you can customize it for your application and I hope I have contributed that knowledge :-)

Happy hacking!

Edit:

You can view comments to this post and a lot more thoughts on HN's ranking here:

<http://news.ycombinator.com/item?id=1781013>

Edit:

Paul Graham has shared the updated [HN ranking algorithm](#):

```
(= gravity* 1.8 timebase* 120 front-threshold* 1
  nourl-factor* .4 lightweight-factor* .17 gag-factor* .1)

(def frontpage-rank (s (o scorefn realscore) (o gravity gravity*)
  (* (/ (let base (- (scorefn s) 1)
            (if (> base 0) (expt base .8) base))
        (expt (/ (+ (item-age s) timebase*) 60) gravity))
     (if (no (in s!type 'story 'poll)) .8
         (blank s!url)               nourl-factor*
         (mem 'bury s!keys)          .001
         (* (contro-factor s)
             (if (mem 'gag s!keys)
                 gag-factor*
                 (lightweight s)
                 lightweight-factor*
                 1))))))
```





Friday, April 6, 2012

## Netflix Recommendations: Beyond the 5 stars (Part 1)

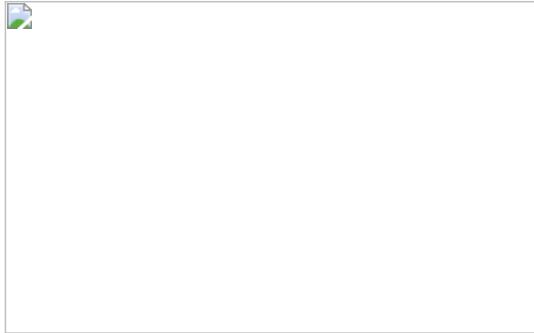
by Xavier Amatriain and Justin Basilico (Personalization Science and Engineering)

In this two-part blog post, we will open the doors of one of the most valued Netflix assets: our recommendation system. In Part 1, we will relate the Netflix Prize to the broader recommendation challenge, outline the external components of our personalized service, and highlight how our task has evolved with the business. In Part 2, we will describe some of the data and models that we use and discuss our approach to algorithmic innovation that combines offline machine learning experimentation with online AB testing. Enjoy... and remember that we are always looking for more star talent to add to our great team, so please take a look at [our jobs page](#).

### The Netflix Prize and the Recommendation Problem

In 2006 we announced the [Netflix Prize](#), a machine learning and data mining competition for movie rating prediction. We offered \$1 million to whoever improved the accuracy of our existing system called *Cinematch* by 10%. We conducted this competition to find new ways to improve the recommendations we provide to our members, which is a key part of our business. However, we had to come up with a proxy question that was easier to evaluate and quantify: the *root mean squared error* (RMSE) of the predicted rating. The race was on to beat our RMSE of 0.9525 with the finish line of reducing it to 0.8572 or less.

A year into the competition, the Korbell team won the first [Progress Prize](#) with an 8.43% improvement. They reported more than 2000 hours of work in order to come up with the final combination of 107 algorithms that gave them this prize. And, they gave us the source code. We looked at the two underlying algorithms with the best performance in the ensemble: *Matrix Factorization* (which the community generally called SVD, *Singular Value Decomposition*) and *Restricted Boltzmann Machines* (RBM). SVD by itself provided a 0.8914 RMSE, while RBM alone provided a competitive but slightly worse 0.8990 RMSE. A linear blend of these two reduced the error to 0.88. To put these algorithms to use, we had to work to overcome some limitations, for instance that they were built to handle 100 million ratings, instead of the more than 5 billion that we have, and that they were not built to adapt as members added more ratings. But once we overcame those challenges, we put the two algorithms into production, where they are still used as part of our recommendation engine.



If you followed the Prize competition, you might be wondering what happened with the final [Grand Prize ensemble](#) that won the \$1M two years later. This is a truly impressive compilation and culmination of years of work, blending hundreds of predictive models to finally cross the finish line. We evaluated some of the new methods offline but the additional accuracy gains that we measured did not seem to justify the engineering effort needed to bring them into a production environment. Also, our focus on improving Netflix personalization had shifted to the next level by then. In the remainder of this post we will explain how and why it has shifted.

### From US DVDs to Global Streaming

One of the reasons our focus in the recommendation algorithms has changed is because Netflix as a whole has changed dramatically in the last few years. Netflix launched an instant streaming service in 2007, one year after the Netflix Prize began. Streaming has not only changed the way our members interact with the service, but also the type of data available to use in our algorithms. For DVDs our goal is to help people fill their queue with titles to receive in the mail over the coming days and weeks; selection is distant in time from viewing, people select carefully because exchanging a DVD for another takes more than a day, and we get no feedback during viewing. For streaming members are looking for something great to watch right now; they can sample a few videos before settling on one, they can consume several in one session, and we can observe viewing statistics such as whether a video was watched fully or only partially.

### Links

- [Netflix US & Canada Blog](#)
- [Netflix America Latina Blog](#)
- [Netflix Brasil Blog](#)
- [Netflix Benelux Blog](#)
- [Netflix DACH Blog](#)
- [Netflix France Blog](#)
- [Netflix Nordics Blog](#)
- [Netflix UK & Ireland Blog](#)
- [Netflix ISP Speed Index](#)
- [Open positions at Netflix](#)
- [Netflix Website](#)
- [Facebook Netflix Page](#)
- [Netflix UI Engineering](#)
- [RSS Feed](#)

### About the Netflix Tech Blog

This is a Netflix blog focused on technology and technology issues. We'll share our perspectives, decisions and challenges regarding the software we build and use to create the Netflix service.

### Blog Archive

- 2015 (14)
- 2014 (37)
- 2013 (52)
- ▼ 2012 (37)
  - December (6)
  - November (3)
  - October (2)
  - September (2)
  - July (6)
  - June (5)
  - May (1)
  - ▼ April (2)
    - Introducing Exhibitor - A Supervisor System for Ap...
  - Netflix Recommendations: Beyond the 5 stars (Part ...)
- March (2)
- February (4)
- January (4)
- 2011 (17)
- 2010 (8)

Another big change was the move from a single website into hundreds of devices. The integration with the Roku player and the Xbox were announced in 2008, two years into the Netflix competition. Just a year later, Netflix streaming made it into the iPhone. Now it is available on a multitude of devices that go from a myriad of [Android devices](#) to the latest [AppleTV](#).

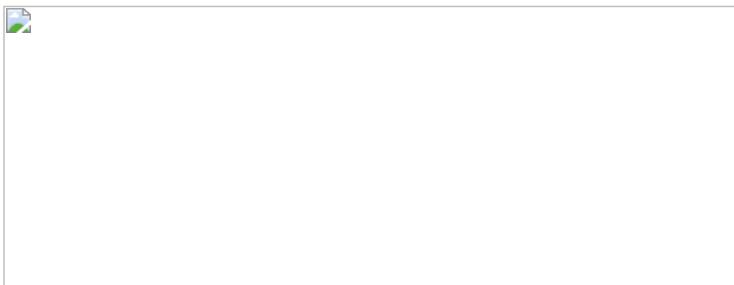
Two years ago, we went international with the [launch in Canada](#). In 2011, we added [43 Latin-American countries](#) and territories to the list. And just recently, we launched in [UK and Ireland](#). Today, Netflix has more than 23 million subscribers in 47 countries. Those subscribers streamed 2 billion hours from hundreds of different devices in the last quarter of 2011. Every day they add 2 million movies and TV shows to the queue and generate 4 million ratings.

We have adapted our personalization algorithms to this new scenario in such a way that now 75% of what people watch is from some sort of recommendation. We reached this point by continuously optimizing the member experience and have measured significant gains in member satisfaction whenever we improved the personalization for our members. Let us now walk you through some of the techniques and approaches that we use to produce these recommendations.

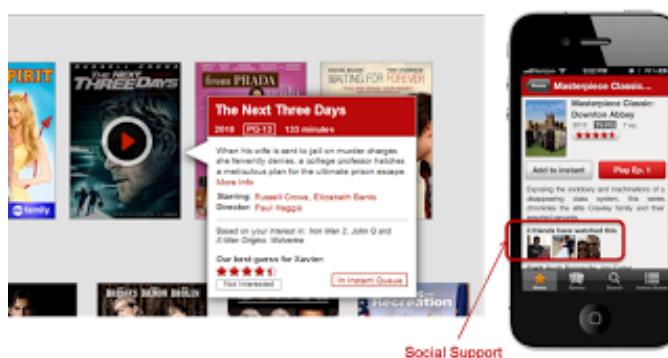
## Everything is a Recommendation

We have discovered through the years that there is tremendous value to our subscribers in incorporating recommendations to personalize as much of Netflix as possible. Personalization starts on our homepage, which consists of groups of videos arranged in horizontal rows. Each row has a title that conveys the intended meaningful connection between the videos in that group. Most of our personalization is based on the way we select rows, how we determine what items to include in them, and in what order to place those items.

Take as a first example the Top 10 row: this is our best guess at the ten titles you are most likely to enjoy. Of course, when we say "you", we really mean everyone in your **household**. It is important to keep in mind that Netflix' personalization is intended to handle a household that is likely to have different people with different tastes. That is why when you see your Top10, you are likely to discover items for dad, mom, the kids, or the whole family. Even for a single person household we want to appeal to your range of interests and moods. To achieve this, in many parts of our system we are not only optimizing for accuracy, but also for **diversity**.



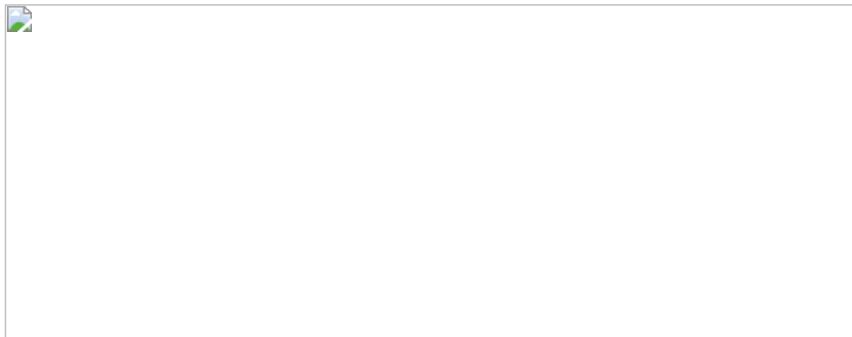
Another important element in Netflix' personalization is **awareness**. We want members to be aware of how we are adapting to their tastes. This not only promotes trust in the system, but encourages members to give feedback that will result in better recommendations. A different way of promoting trust with the personalization component is to provide **explanations** as to why we decide to recommend a given movie or show. We are not recommending it because it suits our business needs, but because it matches the information we have from you: your explicit taste preferences and ratings, your viewing history, or even your friends' recommendations.



On the topic of friends, we [recently released](#) our Facebook connect feature in 46 out of the 47 countries we operate – all but the US because of concerns with the VPPA law. Knowing about your friends not only gives us another signal to use in our personalization algorithms, but it also allows for different rows that rely mostly on your **social** circle to generate recommendations.

## Labels

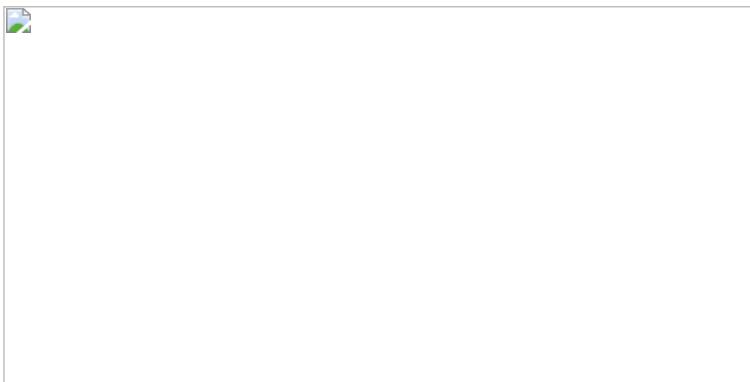
|                                  |
|----------------------------------|
| accelerated compositing (2)      |
| adwords (1)                      |
| Aegisthus (1)                    |
| algorithms (2)                   |
| aminator (1)                     |
| analytics (4)                    |
| Android (1)                      |
| angular (1)                      |
| api (16)                         |
| appender (1)                     |
| Archaius (2)                     |
| architectural design (1)         |
| Asgard (1)                       |
| Astyanax (3)                     |
| authentication (1)               |
| automation (1)                   |
| autoscaling (3)                  |
| availability (4)                 |
| AWS (27)                         |
| benchmark (2)                    |
| big data (10)                    |
| billing (1)                      |
| Blitz4j (1)                      |
| build (3)                        |
| Cable (1)                        |
| caching (1)                      |
| Cassandra (13)                   |
| chaos engineering (1)            |
| chaos monkey (5)                 |
| ci (1)                           |
| classloaders (1)                 |
| Clojure (1)                      |
| cloud (22)                       |
| cloud architecture (15)          |
| cloud prize (3)                  |
| collection (1)                   |
| concurrency (1)                  |
| configuration (2)                |
| configuration management (2)     |
| conformity monkey (1)            |
| content platform engineering (1) |
| continuous delivery (3)          |
| coordination (2)                 |
| cost management (1)              |
| crypto (1)                       |
| Cryptography (2)                 |
| CSS (2)                          |
| CUDA (1)                         |



Some of the most recognizable personalization in our service is the collection of “genre” rows. These range from familiar high-level categories like “Comedies” and “Dramas” to highly tailored slices such as “Imaginative Time Travel Movies from the 1980s”. Each row represents 3 layers of personalization: the choice of genre itself, the subset of titles selected within that genre, and the ranking of those titles. Members connect with these rows so well that we measure an increase in member retention by placing the most tailored rows higher on the page instead of lower. As with other personalization elements, **freshness** and diversity is taken into account when deciding what genres to show from the thousands possible.

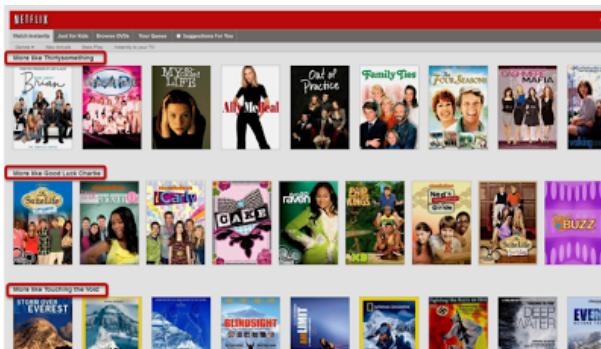


We present an explanation for the choice of rows using a member’s implicit genre preferences – recent plays, ratings, and other interactions --, or explicit feedback provided through our taste preferences survey. We will also invite members to focus a row with additional explicit preference feedback when this is lacking.



**Similarity** is also an important source of personalization in our service. We think of similarity in a very broad sense; it can be between movies or between members, and can be in multiple dimensions such as metadata, ratings, or viewing data. Furthermore, these similarities can be blended and used as features in other models. Similarity is used in multiple contexts, for example in response to a member’s action such as searching or adding a title to the queue. It is also used to generate rows of “adhoc genres” based on similarity to titles that a member has interacted with recently. If you are interested in a more in-depth description of the architecture of the similarity system, you can read about it in [this past post](#) on the blog.

|   |
|---|
| <a href="#">dart (1)</a>                            |
| <a href="#">data migration (1)</a>                  |
| <a href="#">data pipeline (4)</a>                   |
| <a href="#">data science (6)</a>                    |
| <a href="#">data visualization (1)</a>              |
| <a href="#">database (4)</a>                        |
| <a href="#">DataStax (2)</a>                        |
| <a href="#">deadlock (1)</a>                        |
| <a href="#">deep learning (1)</a>                   |
| <a href="#">Denominator (2)</a>                     |
| <a href="#">dependency injection (1)</a>            |
| <a href="#">device (3)</a>                          |
| <a href="#">device proliferation (1)</a>            |
| <a href="#">devops (1)</a>                          |
| <a href="#">distributed (10)</a>                    |
| <a href="#">DNS (1)</a>                             |
| <a href="#">Docker (1)</a>                          |
| <a href="#">Dockerhub (1)</a>                       |
| <a href="#">DSL (1)</a>                             |
| <a href="#">Dyn (1)</a>                             |
| <a href="#">DynECT (1)</a>                          |
| <a href="#">Elastic Load Balancer (1)</a>           |
| <a href="#">elasticsearch (1)</a>                   |
| <a href="#">ELB (1)</a>                             |
| <a href="#">EMR (2)</a>                             |
| <a href="#">encoding (1)</a>                        |
| <a href="#">eucalyptus (1)</a>                      |
| <a href="#">eureka (2)</a>                          |
| <a href="#">evcache (1)</a>                         |
| <a href="#">failover (2)</a>                        |
| <a href="#">fault-tolerance (12)</a>                |
| <a href="#">Flow (1)</a>                            |
| <a href="#">FRP (1)</a>                             |
| <a href="#">functional reactive (1)</a>             |
| <a href="#">garbage (1)</a>                         |
| <a href="#">garbage collection (1)</a>              |
| <a href="#">gc (1)</a>                              |
| <a href="#">Genie (4)</a>                           |
| <a href="#">Governator (1)</a>                      |
| <a href="#">GPU (2)</a>                             |
| <a href="#">Groovy (1)</a>                          |
| <a href="#">Hack Day (2)</a>                        |
| <a href="#">Hadoop (12)</a>                         |
| <a href="#">HBase (1)</a>                           |
| <a href="#">high volume (4)</a>                     |
| <a href="#">high volume distributed systems (8)</a> |
| <a href="#">Hive (2)</a>                            |
| <a href="#">HTML5 (7)</a>                           |
| <a href="#">https (1)</a>                           |



In most of the previous contexts – be it in the Top10 row, the genres, or the similars – **ranking**, the choice of what order to place the items in a row, is critical in providing an effective personalized experience. The goal of our ranking system is to find the best possible ordering of a set of items for a member, within a specific context, in real-time. We decompose ranking into scoring, sorting, and filtering sets of movies for presentation to a member. Our business objective is to maximize member satisfaction and month-to-month subscription retention, which correlates well with maximizing consumption of video content. We therefore optimize our algorithms to give the highest scores to titles that a member is most likely to play and enjoy. Now it is clear that the Netflix Prize objective, accurate prediction of a movie's rating, is just one of the many components of an effective recommendation system that optimizes our members enjoyment. We also need to take into account factors such as context, title popularity, interest, evidence, novelty, diversity, and freshness. Supporting all the different contexts in which we want to make recommendations requires a range of algorithms that are tuned to the needs of those contexts. In the next part of this post, we will talk in more detail about the ranking problem. We will also dive into the data and models that make all the above possible and discuss our approach to innovating in this space.

## On to part 2

Posted by Xavier Amatriain at 6:30 PM

Labels: machine learning, Netflix, personalization, recommendations

Hystrix (5)  
IBM (1)  
ice (1)  
initialization (1)  
innovation (3)  
insights (1)  
inter process communication (1)  
Ipv6 (2)  
isolation (1)  
ISP (1)  
java (4)  
JavaScript (15)  
jclouds (1)  
jenkins (1)  
Karyon (2)  
lifecycle (1)  
lipstick (2)  
load balancing (3)  
locking (1)  
locks (1)  
log4j (1)  
logging (2)  
machine learning (5)  
Map-Reduce (1)  
meetup (3)  
memcache (2)  
memcached (1)  
message security layer (1)  
Mobile (1)  
modules (1)  
monitoring (1)  
msl (1)  
negative keywords (1)  
Netflix (16)  
Netflix API (7)  
netflix graph (1)  
Netflix OSS (11)  
NetflixOSS (12)  
neural networks (1)  
node.js (2)  
NoSQL (5)  
observability (1)  
Open source (9)  
operational excellence (1)  
operational insight (2)  
operational visibility (1)  
optimization (1)  
OSS (2)  
outage (1)

[Newer Post](#)

Home

[Older Post](#)

|                            |
|----------------------------|
| page generation (1)        |
| payments (1)               |
| Paypal (1)                 |
| performance (16)           |
| personalization (4)        |
| phone (1)                  |
| Pig (4)                    |
| Playback (1)               |
| prediction (2)             |
| predictive modeling (2)    |
| Presto (2)                 |
| prize (1)                  |
| pytheas (1)                |
| python (3)                 |
| Quality (1)                |
| rca (2)                    |
| React (1)                  |
| Reactive Programming (2)   |
| real-time insights (1)     |
| Recipe (1)                 |
| recommendations (6)        |
| Redis (2)                  |
| reinvent (2)               |
| reliability (7)            |
| remote procedure calls (1) |
| research (1)               |
| resiliency (7)             |
| REST (2)                   |
| Ribbon (2)                 |
| Riot Games (1)             |
| root-cause analysis (2)    |
| Route53 (1)                |
| rule engine (1)            |
| Rx (1)                     |
| scalability (11)           |
| scale (1)                  |
| scripting library (1)      |
| search (2)                 |
| security (6)               |
| Servo (1)                  |
| shared libraries (1)       |
| simian army (5)            |
| SimpleDB (3)               |
| site reliability (1)       |
| sqoop (1)                  |
| ssd (1)                    |
| ssl (1)                    |
| STAASH (1)                 |
| stream processing (1)      |

|  |
|--|
| <a href="#">streaming</a> (1)            |
| <a href="#">suro</a> (1)                 |
| <a href="#">SWF</a> (1)                  |
| <a href="#">synchronization</a> (1)      |
| <a href="#">tablet</a> (1)               |
| <a href="#">testability</a> (1)          |
| <a href="#">tls</a> (1)                  |
| <a href="#">traffic optimization</a> (1) |
| <a href="#">TV</a> (5)                   |
| <a href="#">UI</a> (12)                  |
| <a href="#">UltraDNS</a> (1)             |
| <a href="#">unit test</a> (2)            |
| <a href="#">uptime</a> (2)               |
| <a href="#">user interface</a> (5)       |
| <a href="#">visualization</a> (1)        |
| <a href="#">WebKit</a> (3)               |
| <a href="#">Wii U</a> (1)                |
| <a href="#">winner</a> (1)               |
| <a href="#">winners</a> (1)              |
| <a href="#">workflow</a> (1)             |
| <a href="#">ZeroToDocker</a> (1)         |
| <a href="#">ZooKeeper</a> (1)            |
| <a href="#">zuul</a> (1)                 |
| <a href="#">"cloud architecture"</a> (3) |

---

Awesome Inc. template. Powered by [Blogger](#).

**Hacker News** new | comments | show | ask | jobs | submit [login](#)

## How Hacker News ranking algorithm works (amix.dk)

311 points by cristoperb 1652 days ago | 74 comments

pg 1652 days ago

That's close to the current version, but a little out of date. Here's the code running now:

```
(= gravity* 1.8 timebase* 120 front-threshold* 1
  nourl-factor* .4 lightweight-factor* .17 gag-factor* .1)

(def frontpage-rank (s (o scorefn realscore) (o gravity gravity*))
  (* (/ (let base (- (scorefn s) 1)
            (if (> base 0) (expt base .8) base))
         (expt (/ (+ (item-age s) timebase*) 60) gravity))
     (if (no (in s!type 'story 'poll)) .8
         (blank s!url)
         (mem 'bury s!keys)
         (* (contro-factor s)
            (if (mem 'gag s!keys)
                gag-factor*
                (lightweight s)
                lightweight-factor*
                1)))))

-----
```

sahillavingia 1652 days ago

For those who aren't well versed in Arc or Lisp, could someone go through the differences?

tel 1652 days ago

Appears to be substantially identical to what's outlined in the post. The only changes are in the penalties dealt out by the type of story, inclusion of a url, buried status, gagged status, and the operation of penalties due to the contro(versial?)-factor function.

If you're just submitting interesting URLs and worry about them being shunted off the main page by gravity then there are no practical differences.

pg 1652 days ago

The gag tag doesn't mean "gagged." It means the post is a gag, in the sense of a joke.

Sorry I can't be more transparent about how contro-factor is calculated, incidentally. Its purpose is to recognize flamewars.

davi 1652 days ago

I'm curious about how you evaluate prospective algorithm changes. Do you roll out a change you think should work and then monitor, or do you have a corpus of e.g. flame wars you test against?

pg 1652 days ago

There's a repl on the server, and I test tweaks on the live site.

techbio 1634 days ago

Do you have a dashboard of some kind to visualize your tests effects? I am curious about the broad kind of insight one could have with such a programmatic access to this community. Perhaps I missed an essay about HN/REPL?

-----  
pypyguy 1652 days ago

Any genius willing to translate this to python?

-----  
sstrudeau 1652 days ago

There's a python implementation in the original article:

```
def calculate_score(votes, item_hour_age, gravity=1.8):
    return (votes - 1) / pow((item_hour_age+2), gravity)
```

-----  
zackattack 1652 days ago

I would prefer PHP but seconded

-----  
sstrudeau 1652 days ago

This is pretty trivial but sure, something like:

```
function calculate_score($votes, $item_hour_age, $gravity=1.8) {
    return ($votes - 1) / pow(($item_hour_age+2), $gravity);
}
```

-----  
mkramlich 1652 days ago

it's interested to see such a direction comparison of PHP to Python.  
PHP is very similar to the Python except with more syntax noise. :)  
And probably worse docs. And worse namespacing, etc. ;)

-----  
EGreg 1652 days ago

do you guys take logs of both sides, so you don't have to keep updating the scores of all the previously posted items? I mean, if all that matters is how big the scores are relative to one another, then  $A > B \Leftrightarrow \log A > \log B$  (since A and B are positive). You'll just have to add a  $G * \log(T+2)$  to the  $\log(P)$  for new items, which can go on for a very long time. ... T is the time since some arbitrary point.

I think I may have just reinvented the way reddit does it :P

-----  
EGreg 1652 days ago

in fact, as someone else said if you then make  $T = \text{the number of submissions / votes since the beginning}$ , people would be able to post at midnight and it would still have a "fair" chance of being ranked along with the other articles posted at busier times.

-----  
antirez 1652 days ago

When I built oknotizie.virgilio.it many years ago, more or less at the same time reddit was created, I used the same base algorithm, that is:  $\text{RANK} = \text{SCORE} / \text{AGE}^{\text{ALPHA}}$ , where ALPHA is the obsolescence factor.

This is a pretty obvious algorithm, but the evil is in the details. First, since oknotizie is based in Italy AGE is calculated in a special way so that nightly hours are calculated in a different way (every hour should be taken into account proportionally to the traffic that there is in this

hour).

Second, there is to do a lot of filtering. Oknotizie is completely built out of anti-spamming: statistical analysis on users voting patterns, cycles detection, an algorithm penalizing similarities in general in the home page, and so forth.

To run a simple HN style site is simple as long as the community is not trying hard to game it. Otherwise it starts to get a much more complex (and sad) affair.

-----

barrkel 1652 days ago

A problem (IMHO) with the HN ranking algorithm is that once a post fails to get traction (perhaps because things were busy at the time it was submitted), it won't really be able to get traction later, even if it's re-discovered 6 hours or 2 days later. Seems to me like velocity ought to be taken into account a little more for items that have otherwise languished.

-----

wensing 1652 days ago

This has been the case with my bootstrapping post. It has been revived a few times thanks to tweets and other references since its initial publication to HN, but it has never risen back to where it was, even though it has twice as many points as it did 11 days ago.

-----

yesbabyyes 1652 days ago

Here's an explanation of other ranking algorithms, including Bayesian average, Wilson score and the ones used on HN, Reddit, StumbleUpon and Del.icio.us:

<http://blog.linkibol.com/2010/05/07/how-to-build-a-popularit...>

-----

jacquesm 1652 days ago

This is not the 'hacker news' ranking algorithm, this is the ranking algorithm distributed with 'ARC', which is the basis for the HN algorithm, but definitely not equal to it.

The biggest missing ingredients are flagged posts dropping off quicker and posts that contain no URL dropping off quicker but there are quite a few other subtle tweaks.

The (very good) reason why the ARC sources do not give out the real ranking algorithm is to make it a bit harder to game the system.

-----

jackowayed 1652 days ago

He glossed over it, but this code does include URL-free posts dropping off faster. See the stuff dealing with "nourl-factor\*". I don't know arc at all, but it appears that having no URL multiplies your final score by a factor of .4, meaning that it's ranked almost 3x lower than it otherwise would be. That surprises me; I've noticed that Ask HN get rated lower, but it doesn't seem that extreme.

So is Hacker News a fork of news.arc, rather than straight news.arc? I figured it was, but never heard that officially (since people refer to news.arc as the HN "source code").

Edit: Also, the "lightweight" thing is interesting. There's something in place that sees if the post is a "rallying cry" or is mostly made of images. Additionally, if you link directly to an image file, or to some list of domains that have been deemed lightweight, that'll get marked as lightweight as well. Lightweight posts have a .3 factor, meaning that they're even more deflated than URL-free posts.

-----

jacquesm 1652 days ago

Ah yes, you're right, the 'nourl' is there, it's in the arc bit, but I couldn't find that in the graphs or in the python code.

-----

amix 1652 days ago

The current HN algorithm probably follows the same basic idea, but the implementation is more complex because of spammers.

I did a fast test of how placements would look like with the vanilla HN algorithm using the current frontpage: <http://paste.plurk.com/show/316811/>

The code to generate the new sorting: <http://paste.plurk.com/show/316812/>

As you can see the rankings are not the same, but very similar...

-----

jacquesm 1652 days ago

Flags are the main factor that you've missed I'm not quite sure how to measure them reliably but from what I've seen in terms of 'jumps' downward on a heavily flagged post it looked like 6 flags will take you off the homepage on to page two when you have about 20 upvotes. Maybe that will allow you to model it.

-----

chehra 1652 days ago

I have two will take you off when you are under 9 upvotes.

-----

J3L2404 1652 days ago

How did you know how many flags it had?

-----

jacquesm 1652 days ago

Counting the jumps down the home page. Would be nice if PG could confirm this either way (right or wrong), but I'm fairly sure that's what it is. It wouldn't make sense any other way (it was this post: <http://news.ycombinator.com/item?id=1759548>) it got to the first spot on the homepage very quickly and then dropped in six steps off the homepage.

19 points, six steps, that's averaging 3 points per step, so it's like a single flag counts for 3 'downvotes' on an article or something close to that.

That's all guesswork of course.

-----

bergie 1652 days ago

I built a reasonably similar ranking system a few years ago, but also taking social media interaction (blog links, delicious bookmarks, etc) with the content items being ranked into account: [http://bergie.iki.fi/blog/calculating\\_news\\_item\\_relevance/](http://bergie.iki.fi/blog/calculating_news_item_relevance/)

You can see it in action on maemo.org: <http://maemo.org/news/>

PHP sources: <http://trac.midgard-project.org/browser/branches/ragnarok/m...>

-----

qeorge 1652 days ago

I made an HN filter for myself, that's basically:

points / comments

It works shockingly well. Its here if anyone would like to check it out:  
<http://www.upthread.com/>

-----

fizx 1652 days ago

Yep, you basically have a controversy filter.

-----

gojomo 1652 days ago

Some weaknesses of this algorithm are:

- (1) Wall-clock hours penalize an article even if no one is reading (overnight, for example). A time denominated in ticks of actual activity (such as views of the 'new' page, or even upvotes-to-all-submissions) might address this.
- (2) An article that misses its audience first time through -- perhaps due to (1) or a bad headline -- may never recover, even with a later flurry of votes far beyond what new submissions are getting.

Without checking the exact numbers, consider a contrived example: Article A is submitted at midnight and 3 votes trickle in until 8am. Then at 8am article B is submitted. Over the next hour, B gets 6 votes and A gets 9 votes. (Perhaps many of those are duplicate-submissions that get turned into upvotes.) A has double the total votes, and 50% more votes even in the shared hour, but still may never rank above B, because of the drag of its first 8 hours.

(I think you'd need to timestamp each vote for an improved decay function.)

-----

angusgr 1652 days ago

*Wall-clock hours penalize an article even if no one is reading (overnight, for example)*

I'd be interested to know what the hourly fluctuation for HN is actually like, on account of having readers all over the world.

I'm in Australia, so your example "submitted at midnight" California time[1] means submitted at 6pm my time. Also 8am London time, 11am Moscow time. :).

[1] I'm going to go ahead and assume you're in California. ;)

-----

gojomo 1652 days ago

I'm in California, usually, but have often observed HN through the California night -- either because of my own odd online hours, or trips to distant time-zones.

It's true there's never total quiescence, but the pace of actions changes by a noticeable factor. (Without going to the data, I'd guess 5X from trough to peak over a day's cycle, and a somewhat smaller weekend-to-weekday difference. Holidays and nice bay area weather also play a factor.)

-----

ig1 1652 days ago

I've found optimal submission time to generally be midday london time, you catch the european lunch-time traffic and the US wake-up/get-into-work traffic. I don't think traffic from anywhere else is heavy enough to matter.

-----

noahc 1652 days ago

I'm in Iowa, so it's central time. But it's not uncommon to see 2 or 3 hour old stories on the front page by 10:30 or so.

-----

fizx 1652 days ago

I believe reddit has historically done something similar. Rather than having a given article's ranking decay over time, they simply hand out better ranks to newer articles, leading to continual inflation. A naive example of this sort of ranking would be (timestamp in hours + upvotes - downvotes).

This would make your (excellent) idea easy to implement, because you could just use an autoincrement key as the timestamp, ignoring any sort of decay calculations.

-----

jasonwatkinspdx 1651 days ago

This inflation based approach also plays nicely with database indexes.

pmarin 1652 days ago

(1) "overnight" not exist in the Internet era. (from Spain)

-----  
cryptoz 1652 days ago

You are right of course, but I suspect the vast majority of HN users are not only in the USA but specifically in California. So even though people are using the site around the clock, there is probably a lot more traffic during the "awake hours" of California.

-----  
BrandonM 1652 days ago

> I suspect the vast majority of HN users are not only in the USA but specifically in California

I suspect that you have a nonstandard definition of "vast majority." I'd be surprised if even 1/4 of HN users are in California. There's a whole wide world out there! :)

-----  
duck 1652 days ago

Although my Hacker Newsletter stats could be completely different than HN, it is a subset of HN users so it might be relevant - California averages around 20% of my opens and the USA about 70% of the total.

Edit: Just to add to this - New York and Massachusetts are the next highest and combined make up about the same as California.

-----  
mkramlich 1652 days ago

Yes and when somebody in France makes a French equivalent of Le Hacker News, and a bunch of Americans start posting there, in second language French, then the "host country" readers can make the same geo assumption in reverse. ;)

-----  
templaedhel 1652 days ago

25% of a website's userbase being located in one state of one country is a "vast majority" on the Internet, it having a far higher HN'er/population ratio than any other state or country.

-----  
rottencupcakes 1652 days ago

To be pedantic, the definition of majority is half the group, and 'vast' implies upwards of 70%.

You probably mean something like a "large plurality"

-----  
amix 1652 days ago

The thing I like about HN's ranking algorithm is how simple it is. Implementing something more sophisticated would require more work and probably much better servers.

Recently, I have been implementing a ranking algorithm and I started in the wrong direction by taking all kinds of things into consideration. This is dangerous because you spend a lot of time on something that is mostly irrelevant.

The bottom line is that HN's algorithm works pretty well for the majority of cases. There are some edges, but solving them would not be trivial and would require a lot more work.

gojomo 1652 days ago

I'd agree simple is better.

Of the two suggestions, replacing hours with artificial ticks adds very little complexity: it's the same formula, with one value replaced, and the accompanying factor adjusted. (The ticks might be submission-count, upvote-count, visit-count, or anything else trivially tallied -- it may not make much difference, except that over time greater activity could require adjusting the tick-deflator-factor.)

jshen 1652 days ago

visit count is not trivial. A db write for every page view is heavy.

seabee 1652 days ago

Persist it in memory, you only need periodical writes for something like that. We're not running on a \$5 powweb account here.

jshen 1650 days ago

Right, but then it's not trivial which was my point. It can. Be trivial and not scale, or scale and not be trivial.

I'm not sure why I get down voted for this.

eculver 1652 days ago

I do agree that that the *Wall-clock* hours argument is somewhat weak due to the international crowd behind HN, but it would be interesting to see how weekends or really more general traffic patterns such as what may happen on holidays attribute to scores. I know that in these cases, I generally diverge from my normal HN-checking patterns.

sesqu 1651 days ago

(*I think you'd need to timestamp each vote for an improved decay function.*)

Well, somewhat. You'd need a timestamped sufficient statistic, but not necessarily each vote.

Consider ACO. Given +1:

```
t0=last_change_timestamp  
s0=score_after_last_change  
s1=1+s0/exp(t1-t0)
```

ma2rten 1652 days ago

HN doesn't really experience a night, since people from all over the planet or on the site. But maybe it would be better to divide by a number of impressions weighted with something.

EGreg 1652 days ago

Well, you can fix (1) by simply incrementing T not by actual time but just be the total # of votes/submissions since the beginning.

DeusExMachina 1652 days ago

Does anybody care to explain the strange indentation I see in the Arc code? I know Lisp a

little (mostly Clojure), but I don't get the indentation of the code at the bottom of the algorithm where it looks like it's branching in two parts. Is this peculiar to Arc? Or to other Lisps as well?

rntz 1652 days ago

That's because of the way Arc does if-expressions. In Scheme and Common Lisp, you have two conditional forms: 'if and 'cond. 'if takes three arguments (or two, with an implied nil, in CL), and is analogous to an if-then-else in other languages:

```
> (if #t 'then 'else)
  then
> (if #f 'then 'else)
  else
```

'cond, in contrast, takes as many branches as you like, but they're parenthesized like so:

```
> (cond (#f 'a)
        (#t (display "I can have a body here!\n")
             'b)
        (#t 'c))
I can have a body here!
b
```

In arc, there's just 'if, which is like 'cond with a lot of implicit parenthesization and else-branches:

```
arc> (if t 'then 'else)
  then
arc> (if nil 'then)           ; if no else-branch is given, nil is implied
  nil
arc> (if nil 'a
        t   (do (prn "'do is like Scheme's 'begin or CL's 'progn.")
              'b)
        'else)
'do is like Scheme's 'begin or CL's 'progn.
  b
```

pmjordan 1652 days ago

I've barely looked at Arc[1] but as far as I remember, the *if* macro/special form allows an arbitrary number of argument forms, a bit like a combination of Clojure/CL's *if* and *cond*:

```
(if
  cond1  expr1
  cond2  expr2
  ..
  condN  exprN
  else-expr)
```

With an even number of arguments, there is no else-expr (the same as CL/Clojure *cond*).

The indentation is probably to distinguish between condition and conditional expression - the second column is what could be evaluated and returned from the whole expression.

[1] I too use Clojure

twism 1652 days ago

Doesn't stray that much from clojure, so I think it's a lisp thing. To keep bindings in clojure with different length names more readable, people tend to indent like this:

```
(let [really-long-name 1
```

```

score          (get-score x)
position      (get-pos) x
test          test]
(...           ....))))))
```

Sort of the same effect in the arc snippet in OP.

-----  
kens 1652 days ago

For more details on the algorithm, see my article "Inside the news.yc ranking formula" from last year: <http://www.arcfn.com/2009/06/how-does-newsyc-ranking-work.ht...>

-----  
johns 1652 days ago

The home page algorithm seems to have changed recently, but the RSS feed hasn't and is much noisier. It would be nice to see the RSS feed updated to reflect the home page changes (or confirmation that I'm just perceiving a difference that doesn't actually exist).

-----  
callmeed 1652 days ago

So, if you're implementing this in a framework like Django or Rails, can you get a result set in this order directly from a query? Or do you have to query then sort?

-----  
endtime 1652 days ago

In Django, you could make the ranking score a property of the model and then do `query.order_by('-score')`.

-----  
tamersalama 1652 days ago

A basic question: If this was performed on page load, and in-memory, how would the first db fetch occur? Unless this is pushed 'somehow' to the database.

-----  
gsivil 1652 days ago

Nice post. Do you know what is the algorithm for the ranking of comments? I think this would be interesting to write about that too.

-----  
brianbreslin 1652 days ago

so this means you'd be best served to submit something at or nearest peak hours?

what are the peak use hours on HN? since everyone is a hacker, i'd assume it was evening hours on east and west coast US as heaviest load? not 9-5 ET?

-----  
pavel\_lishin 1652 days ago

Don't forget employed people with ADD, or people who are bored at work, who stroll over here when they need a break from their job.

-----  
b\_emery 1652 days ago

Looks like there is a built in max lifetime of about 5-10 hrs.

-----  
seldo 1652 days ago

I feel like lots of big stories have managed to last overnight, so at least 12 hours, so I don't think this can be true. I've no idea how I would prove that though.

thiele 1652 days ago

This is also anecdotal but I think TechCrunch's "AngelGate" article was on the front page for almost 3 days.

-----  
b\_emery 1652 days ago

Maybe 'half life' is a better way to put it. Looking at the graphs, the score drops to ~1 in 5-10 hrs, unless the points are very high.

-----  
noarchy 1652 days ago

I've wondered if there aren't some hands-on efforts (as in, non-automated) to keep particular stories on the front page.

-----  
d0m 1652 days ago

I thought the "secret sauce" was hidden from Arc sources..?

-----  
tptacek 1652 days ago

My understanding is that the actual HN implementation is heavily customized and not public.

-----  
grourk 1652 days ago

Maybe what's hidden is the code for determining what counts as a vote. e.g., multiple votes from the same IP may be discounted or simply counted as a single vote. Or a bunch of votes coming from fresh accounts created at or after submission time.

-----  
weaksausage 1652 days ago

That algorithm has been in the source for a while. What is unknown is the current constants and if he changed it at all since first publishing it.

-----  
10smom 1652 days ago

Thanks for the info! now I need to get my son to explain to me. :)

[Guidelines](#) | [FAQ](#) | [Support](#) | [API](#) | [Lists](#) | [Bookmarklet](#) | [DMCA](#) | [Y Combinator](#) | [Apply](#) | [Contact](#)

Search:

*Evanmiller.org*

# How Not To Sort By Average Rating

By [Evan Miller](#)

February 6, 2009 ([Changes](#))

**PROBLEM:** You are a web programmer. You have users. Your users rate stuff on your site. You want to put the highest-rated stuff at the top and lowest-rated at the bottom. You need some sort of "score" to sort by.

**WRONG SOLUTION #1:** Score = (Positive ratings) - (Negative ratings)

*Why it is wrong:* Suppose one item has 600 positive ratings and 400 negative ratings: 60% positive. Suppose item two has 5,500 positive ratings and 4,500 negative ratings: 55% positive. This algorithm puts item two (score = 1000, but only 55% positive) above item one (score = 200, and 60% positive). **WRONG.**

Sites that make this mistake: Urban Dictionary

The screenshot shows two entries from the Urban Dictionary:

- 2. normal** 209 up, 50 down   
 A word made up by this corrupt society so they could single out and attack those who are different  
*Normal is nothing but a word made up by society*  
 conformists worker bees in crowd followers mindless  
 by Bill Oct 6, 2005 share this add comment
- 3. normal** 118 up, 25 down

**WRONG SOLUTION #2:** Score = Average rating = (Positive ratings) / (Total ratings)

*Why it is wrong:* Average rating works fine if you always have a ton of ratings, but suppose item 1 has 2 positive ratings and 0 negative ratings. Suppose item 2 has 100 positive ratings and 1 negative rating. This algorithm puts item two (tons of positive ratings) below item one (very few positive ratings). **WRONG.**

Sites that make this mistake: Amazon.com

13.



**SALTON HOUSEWARES, INC.  
TR2500C ULTIMATE PLUS  
BREAKMAKER**  
Buy new: \$135.99  
In Stock  
★★★★★ (1)

14.



**KitchenAid KP26M1XLC  
Professional 600 Series 6-Quart  
Stand Mixer, Licorice**  
Buy new: \$499.99 \$329.99  
10 Used & new from \$325.00  
Get it by **Monday, Feb 9** if you order in  
the next **19 hours** and choose one-day  
shipping.  
Eligible for **FREE** Super Saver Shipping.  
★★★★★ (580)

**CORRECT SOLUTION:** Score = Lower bound of Wilson score confidence interval for a Bernoulli parameter

*Say what:* We need to balance the proportion of positive ratings with the uncertainty of a small number of observations. Fortunately, the math for this was worked out in 1927 by Edwin B. Wilson. What we want to ask is: *Given the ratings I have, there is a 95% chance that the "real" fraction of positive rating is at least what?* Wilson gives the answer. Considering only positive and negative ratings (i.e. not a 5-star scale), the lower bound on the proportion of positive ratings is given by:

$$\left( \hat{p} + \frac{z_{\alpha/2}^2}{2n} \pm z_{\alpha/2} \sqrt{\frac{[\hat{p}(1 - \hat{p}) + z_{\alpha/2}^2/4n]/n}{1 + z_{\alpha/2}^2/n}} \right) / (1 + z_{\alpha/2}^2/n).$$

(Use minus where it says plus/minus to calculate the lower bound.) Here  $\hat{p}$  is the *observed* fraction of positive ratings,  $z_{\alpha/2}$  is the  $(1-\alpha/2)$  quantile of the standard normal distribution, and  $n$  is the total number of ratings. The same formula implemented in Ruby:

```
require 'statistics2'

def ci_lower_bound(pos, n, confidence)
  if n == 0
    return 0
  end
  z = Statistics2.pnormaldist(1-(1-confidence)/2)
  phat = 1.0*pos/n
  (phat + z*z/(2*n) - z * Math.sqrt((phat*(1-phat)+z*z/(4*n))/n))/(1+z*z/n)
end
```

`pos` is the number of positive ratings, `n` is the total number of ratings, and `confidence` refers to the statistical confidence level: pick 0.95 to have a 95% chance that your lower bound is correct, 0.975 to have a 97.5% chance, etc. The z-score in this function never changes, so if you don't have a statistics

package handy or if performance is an issue you can always hard-code a value here for  $z$ . (Use 1.96 for a confidence level of 0.95.)

---

**UPDATE, April 2012:** Here is an illustrative SQL statement that will do the trick, assuming you have a `widgets` table with positive and negative ratings, and you want to sort them on the lower bound of a 95% confidence interval:

```
SELECT widget_id, ((positive + 1.9208) / (positive + negative) -  
    1.96 * SQRT((positive * negative) / (positive + negative) + 0.9604) /  
    (positive + negative)) / (1 + 3.8416 / (positive + negative))  
AS ci_lower_bound FROM widgets WHERE positive + negative > 0  
ORDER BY ci_lower_bound DESC;
```

If your boss doesn't believe that such a complicated SQL statement could possibly return a useful result, just compare the results to the other two method described above:

```
SELECT widget_id, (positive - negative)  
AS net_positive_ratings FROM widgets ORDER BY net_positive_ratings DESC;  
  
SELECT widget_id, positive / (positive + negative)  
AS average_rating FROM widgets ORDER BY average_rating DESC;
```

You will quickly see that the extra bit of math makes all the good stuff bubble up to the top. (But before running this SQL on a massive database, talk to your friendly neighborhood database administrator about proper use of indexes.)

---

I initially devised this method for a Chuck Norris-style fact generator to honor of one of my professors, but it has since caught on at places like [Reddit](#), [Yelp](#), and [Digg](#).

## OTHER APPLICATIONS

The Wilson score confidence interval isn't just for sorting, of course. It is useful whenever you want to know with confidence what percentage of people took some sort of action. For example, it could be used to:

- Detect spam/abuse: What percentage of people who see this item will mark it as spam?
- Create a "best of" list: What percentage of people who see this item will mark it as "best of"?
- Create a "Most emailed" list: What percentage of people who see this page will click "Email"?

Indeed, it may be more useful in a "top rated" list to display those items with the highest number of positive ratings *per page view, download, or purchase*, rather than positive ratings per rating. Many people who find something mediocre will not bother to rate it at all; the act of viewing or purchasing something and declining to rate it contains useful information about that item's quality.

## CHANGES

- Apr. 4, 2012: New SQL implementation
- Nov. 13, 2011: Fixed statistical confidence language and altered code example accordingly

- Feb. 15: Clarified the statistical power example
- Feb. 13 II: "Other applications"
- Feb. 13: General clarification, plus a link to the relevant Wikipedia article.
- Feb. 12, 2009: The example in "Wrong Solution #1" was erroneous. It has been fixed.

## REFERENCES

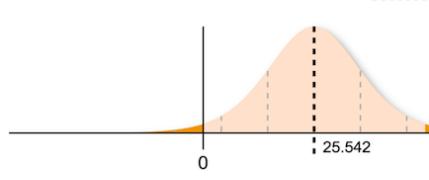
[Binomial proportion confidence interval \(Wikipedia\)](#)

Agresti, Alan and Brent A. Coull (1998), "Approximate is Better than 'Exact' for Interval Estimation of Binomial Proportions," *The American Statistician*, 52, 119-126.

Wilson, E. B. (1927), "Probable Inference, the Law of Succession, and Statistical Inference," *Journal of the American Statistical Association*, 22, 209-212.

You're reading [evanmiller.org](#), a random collection of math, tech, and musings. For a Bayesian perspective on average ratings, check out my other articles, [Bayesian Average Ratings](#) and [Ranking Items With Star Ratings](#).

If you run A/B tests frequently, be sure to check out my collection of [Awesome A/B Tools](#):

|  |  |  |
|--|--|--|
| <b>Answer:</b><br><b>1,005</b><br>per branch<br><br><a href="#">Sample Size Calculator</a> | <b>Confidence interval</b><br><br><a href="#">Chi-Squared Test</a> | <b>Difference of means</b><br>$d = 25.542 \quad SE = 10$<br><br><a href="#">Two-Sample T-Test</a> |
|--|--|--|

Finally, if you own a Mac, my desktop statistics software [Wizard](#) can help you analyze **more data in less time** and **communicate discoveries visually** without spending days struggling with pointless command syntax. Check it out!



**Wizard**  
Statistical analyzer

[Back to Evan Miller's home page](#) – [Follow on Twitter](#) – [Subscribe to RSS](#)

Hacking and Gonzo, a publication by Amir Salihefendic since 2000

[Search >](#)

# How Reddit ranking algorithms work

This is a follow up post to [How Hacker News ranking algorithm works](#). This time around I will examine how [Reddit's](#) default story and comment rankings work. Reddit's algorithms are fairly simple to understand and to implement and in this post I'll dig deeper into them.



The first part of this post will focus on story ranking, i.e. how are Reddit stories ranked? The second part of this post will focus on comment ranking, which does not use the same ranking as stories (unlike [Hacker News](#)), Reddit's comment ranking algorithm is quite interesting and the idea guy behind it is Randall Munroe (the author of [xkcd](#)).



I'm Amir Salihefendic founder and CEO of [Doist](#)

## Stay in Touch

[Twitter @amix3k](#)

[amix@amix.dk](#)

[Blog RSS feed](#)

## Current Focus

[Doist](#)

[Todoist](#)

[Wedoist](#)

[github.com/amix](#)

## Labels

[AJAX and comet](#)

[amix.dk related](#)

[Announcements](#)

[Benchmarks](#)

[Books](#)

[Code](#)

[Code improvement](#)

[Code rewrite](#)

[Database](#)

[Design](#)

[Education](#)

[Interesting](#)

[JavaScript](#)

[Life](#)

## Digging into the story ranking code

Reddit is open sourced and the code is freely available. Reddit is implemented in Python and their [code is located here](#). Their sorting algorithms are implemented in [Pyrex](#), which is a language to write Python C extensions. They have used Pyrex for speed reasons. I have rewritten [their Pyrex implementation](#) into pure Python since it's easier to read.

The default story algorithm called the hot ranking is implemented like this:

```
#Rewritten code from /r2/r2/lib/db/_sorts.pyx

from datetime import datetime, timedelta
from math import log

epoch = datetime(1970, 1, 1)

def epoch_seconds(date):
    """Returns the number of seconds from the epoch to date."""
    td = date - epoch
    return td.days * 86400 + td.seconds + (float(td.microseconds) / 1000000)

def score(ups, downs):
    return ups - downs

def hot(ups, downs, date):
    """The hot formula. Should match the equivalent function in postgres."""
    s = score(ups, downs)
    order = log(max(abs(s), 1), 10)
    sign = 1 if s > 0 else -1 if s < 0 else 0
    seconds = epoch_seconds(date) - 1134028003
    return round(sign * order + seconds / 45000, 7)
```

In mathematical notation the hot algorithm looks like this (I have this from [SEOMoz](#), but I doubt they are the author of this):

Given the time the entry was posted  $A$  and the time of 7:46:43 a.m. December 8, 2005  $B$ , we have  $t_s$  as their difference in seconds

$$t_s = A - B$$

and  $x$  as the difference between the number of up votes  $U$  and the number of down votes  $D$

$$x = U - D$$

where  $y \in \{-1, 0, 1\}$

$$y = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

and  $z$  as the maximal value, of the absolute value of  $x$  and 1

$$z = \begin{cases} |x| & \text{if } |x| \geq 1 \\ 1 & \text{if } |x| < 1 \end{cases}$$

we have the rating as a function  $f(t_s, y, z)$

$$f(t_s, y, z) = \log_{10} z + \frac{yt_s}{45000}$$

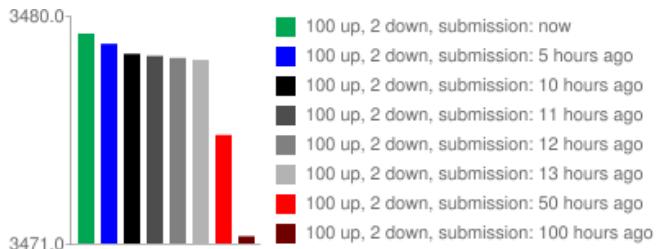
## Effects of submission time

Following things can be said about submission time related to story ranking:

Submission time has a big impact on the ranking and the algorithm will rank newer stories higher than older

The score won't decrease as time goes by, but newer stories will get a higher score than older. This is a different approach than the Hacker News's algorithm which decreases the score as time goes by

Here is a visualization of the score for a story that has same amount of up and downvotes, but different submission time:



## The logarithm scale

Reddit's hot ranking uses the logarithm function to weight the first votes higher than the rest. Generally this applies:

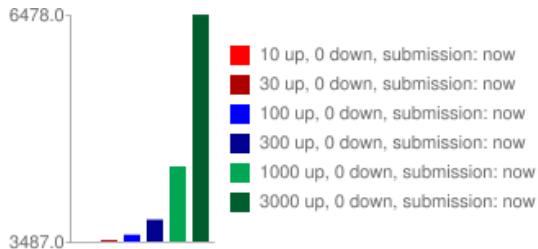
The first 10 upvotes have the same weight as the next 100 upvotes which have the same weight as the next 1000 etc...

Here is a visualization:

- [node.js](#)
- [Orangoo](#)
- [Plurk](#)
- [Political](#)
- [Posters](#)
- [Presentations](#)
- [Productivity](#)
- [Psychology](#)
- [Python](#)
- [Reading list](#)
- [Security](#)
- [Skeletonz](#)
- [Stuff](#)
- [Tips](#)
- [Todoist](#)
- [VIM Editor](#)
- [Wedoist](#)
- [Archive](#)
- [2015](#)
- [2014](#)
- [2013](#)
- [2012](#)
- [2011](#)
- [2010](#)
- [2009](#)
- [2008](#)
- [2007](#)
- [2006](#)
- [2005](#)
- [2004](#)



Without using the logarithm scale the score would look like this:

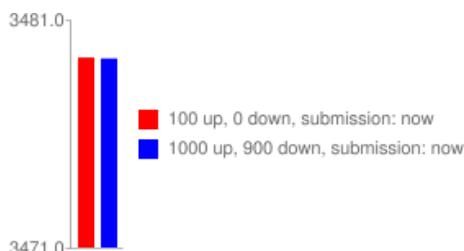


## Effects of downvotes

Reddit is one of the few sites that has downvotes. As you can read in the code a story's "score" is defined to be:

`up_votes - down_votes`

The meaning of this can be visualized like this:



This has a big impact for stories that get a lot of upvotes and downvotes (e.g. controversial stories) as they will get a lower ranking than stories that just get upvotes. This could explain why kittens (and other non-controversial stories) rank so high :)

## Conclusion of Reddit's story ranking

Submission time is a very important parameter, generally newer stories will rank higher than older

The first 10 upvotes count as high as the next 100. E.g. a story that has 10 upvotes and a story that has 50 upvotes will have a similar ranking

Controversial stories that get similar amounts of upvotes and downvotes will get a low ranking compared to stories that mainly get upvotes

## How Reddit's comment ranking works

Randall Munroe of xkcd is the idea guy behind Reddit's best ranking. He has written a great blog post about it:

[reddit's new comment sorting system](#)

You should read his blog post as it explains the algorithm in a very understandable way. The outline of his blog post is following:

Using the hot algorithm for comments isn't that smart since it seems to be heavily biased toward comments posted early

In a comment system you want to rank the best comments highest regardless of their submission time

A solution for this has been found in 1927 by Edwin B. Wilson and it's called "Wilson score interval", Wilson's score interval can be made into "the confidence sort"

The confidence sort treats the vote count as a statistical sampling of a hypothetical full vote by everyone - like in an opinion poll

[How Not To Sort By Average Rating](#) outlines the confidence ranking in higher detail, definitely recommended reading!

## Digging into the comment ranking code

The confidence sort algorithm is implemented in `_sorts.pyx`, I have rewritten their Pyrex implementation into pure Python (do also note that I have removed their caching optimization):

```
#Rewritten code from /r2/r2/lib/db/_sorts.pyx

from math import sqrt

def _confidence(ups, downs):
    n = ups + downs

    if n == 0:
        return 0

    z = 1.0 #1.0 = 85%, 1.6 = 95%
    phat = float(ups) / n
    return sqrt(phat+z*z/(2*n)-z*((phat*(1-phat)+z*z/(4*n))/n))/(1+z*z/n)

def confidence(ups, downs):
    if ups + downs == 0:
        return 0
    else:
        return _confidence(ups, downs)
```

The confidence sort uses [Wilson score interval](#) and the mathematical notation looks like this:

$$\frac{\hat{p} + \frac{1}{2n}z_{1-\alpha/2}^2 \pm z_{1-\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z_{1-\alpha/2}^2}{4n^2}}}{1 + \frac{1}{n}z_{1-\alpha/2}^2}$$

In the above formula the parameters are defined in a following way:

$\hat{p}$  is the observed fraction of positive ratings

$n$  is the total number of ratings

$z_{\alpha/2}$  is the  $(1-\alpha/2)$  quantile of the standard normal distribution

Let's summarize the above in a following manner:

The confidence sort treats the vote count as a statistical sampling of a

hypothetical full vote by everyone

The confidence sort gives a comment a provisional ranking that it is 85% sure it will get to

The more votes, the closer the 85% confidence score gets to the actual score

Wilson's interval has good properties for a small number of trials and/or an extreme probability

Randall has a great example of how the confidence sort ranks comments [in his blog post](#):

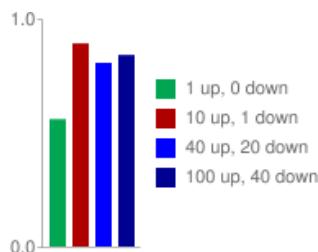
If a comment has one upvote and zero downvotes, it has a 100% upvote rate, but since there's not very much data, the system will keep it near the bottom. But if it has 10 upvotes and only 1 downvote, the system might have enough confidence to place it above something with 40 upvotes and 20 downvotes -- figuring that by the time it's also gotten 40 upvotes, it's almost certain it will have fewer than 20 downvotes. And the best part is that if it's wrong (which it is 15% of the time), it will quickly get more data, since the comment with less data is near the top.

## Effects of submission time: there are none!

The great thing about the confidence sort is that submission time is irrelevant (much unlike the hot sort or Hacker News's ranking algorithm). Comments are ranked by confidence and by data sampling - - i.e. the more votes a comment gets the more accurate its score will become.

## Visualization

Let's visualize the confidence sort and see how it ranks comments. We can use Randall's example:



As you can see the confidence sort does not care about how many votes a comment have received, but about how many upvotes it has compared to the total number of votes and to the sampling size!

## Application outside of ranking

Like [Evan Miller](#) notes Wilson's score interval has applications outside of ranking. He lists 3 examples:

Detect spam/abuse: What percentage of people who see this item will mark it as spam?

Create a "best of" list: What percentage of people who see this item will mark it as "best of"?

Create a "Most emailed" list: What percentage of people who see this page will click "Email"?

To use it you only need two things:

- the total number of ratings/samplings
- the positive number of ratings/samplings

Given how powerful and simple this is, it's amazing that most sites today use the naive ways to rank their content. This includes billion dollar companies like [Amazon.com](#), which define Average rating = (Positive ratings) / (Total ratings).

## Conclusion

I hope you have found this useful and leave comments if you have any questions or remarks.

Happy hacking as always :)

## Related

[Reddit's comment ranking algorithm](#), discusses a bug that Reddit's implementation has

23. Nov 2010 • [Code](#) · [Design](#) · [Python](#)

# Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems

Badrul Sarwar<sup>†\*</sup>, George Karypis<sup>‡</sup>, Joseph Konstan<sup>†</sup>, and John Riedl<sup>†</sup>

{sarwar, karypis, konstan, riedl}@cs.umn.edu

<sup>†</sup>GroupLens Research Group / <sup>‡</sup>Army HPC Research Center

Department of Computer Science and Engineering

University of Minnesota, Minneapolis, MN 55455, USA

## Abstract

We investigate the use of dimensionality reduction to improve the performance for a new class of data analysis software called “recommender systems”. Recommender systems apply knowledge discovery techniques to the problem of making personalized product recommendations during a live customer interaction. The tremendous growth of customers and products in recent years poses some key challenges for recommender systems. These are: producing high quality recommendations and performing many recommendations per second for millions of customers and products. Singular Value Decomposition(SVD)-based recommendation algorithms can quickly produce high quality recommendations, but has to undergo very expensive matrix factorization steps. In this paper, we propose and experimentally validate a technique that has the potential to incrementally build SVD-based models and promises to make the recommender systems highly scalable.

## 1 Introduction

Recommender systems have evolved in the extremely interactive environment of the Web. They apply data analysis techniques to the problem of helping customers find which products they would like to purchase at E-commerce sites. These systems, especially the collaborative filtering based ones [3, 5, 7, 8, 11], are rapidly becoming a crucial tool in E-commerce on the Web. Nowadays, they are being stressed by the huge volume of customer data in existing corporate databases, and will be stressed even more in future by the increasing volume of customer data available on the Web. The tremendous growth of customers and products poses two key challenges for recommender systems. The first challenge is to improve the quality of the recommendations for the consumers. Consumers need recommendations they can trust to help them find products they will like. If a consumer trusts a recommender system,

purchases a product, and finds out he does not like the product, the consumer will be unlikely to use the recommender system again. Another challenge is to improve the scalability of the collaborative filtering algorithms. These algorithms are able to search tens of thousands of potential neighbors in real-time, but the demands of modern E-commerce systems are to search tens of millions of potential neighbors.

In some ways these two challenges are in conflict, since the less time an algorithm spends searching for neighbors, the more scalable it will be, and the worse its quality. For this reason, it is important to treat the two challenges simultaneously so the solutions discovered are both useful and practical. New technologies are needed that can dramatically improve the scalability of recommender systems. Researchers [1, 4, 9, 10] suggest that Singular Value Decomposition (SVD) may be such a technology in some cases. SVD-based approach produced results that were better than a traditional collaborative filtering algorithm most of the time when applied to a Movie data set [9]. However, SVD-based recommender systems suffer one serious limitation that makes them less suitable for large-scale deployment in E-commerce. The matrix factorization step associated with these systems is computationally very expensive and is a major stumbling block towards achieving high scalability.

In this paper, we experiment with an incremental model-building technique for generating SVD-based recommendations that has the promise of being highly scalable while producing good predictive accuracy. In particular, we present an algorithm that builds upon a small pre-computed SVD model and provides larger SVD models using inexpensive techniques. Our experimental results suggest that the overall algorithm works twice as fast while producing similar prediction accuracy.

The rest of the paper is organized as follows. The next section gives a brief overview of the SVD-based prediction generation process and discusses its promises and challenges. Section 3 outlines our incremental SVD algorithm. Section 4 presents our experimental procedure, results and discussion. The final

\*Currently with the Computer Science Department, San Jose State University, San Jose, CA 95112, USA. Email: sarwar@cs.sjsu.edu, Phone: +1 408-245-8202

section provides some concluding remarks and future research directions.

## 2 Dimensionality Reduction for Collaborative Filtering

In this section we briefly discuss how one dimensionality reduction technique can potentially be used for prediction generation. We then present some of the challenges of such algorithms and propose an incremental technique to make them highly scalable.

### 2.1 Promises of Dimensionality Reduction

The goal of CF-based recommendation algorithms [7, 8, 11] is to suggest new products or to predict the utility of a certain product for a particular customer, based on the customer’s previous liking and the opinions of other like-minded customers. These systems have been successful in several domains. However, in our earlier papers [9, 10], we mentioned some limitations of these systems, namely *sparsity*, *scalability*, and *synonymy*. The weakness of CF algorithms for large, sparse databases led us to explore alternative recommender system algorithms. After reviewing several techniques, we decided to try applying Latent Semantic Indexing (LSI) to reduce the dimensionality of our customer-product ratings matrix. LSI is a dimensionality reduction technique that has been widely used in information retrieval (IR) to solve the problems of synonymy and polysemy [2]. LSI, which uses *singular value decomposition (SVD)* as its underlying dimensionality reduction algorithm, maps nicely into the collaborative filtering recommender algorithm challenge.

**Singular Value Decomposition (SVD).** SVD is a matrix factorization technique commonly used for producing *low-rank* approximations. Given an  $m \times n$  matrix  $A$ , with rank  $r$ , the singular value decomposition,  $SVD(A)$ , is defined as

$$SVD(A) = U \times S \times V^T \quad (1)$$

Where  $U$ ,  $S$  and  $V$  are of dimensions  $m \times m$ ,  $m \times n$ , and  $n \times n$ , respectively. Matrix  $S$  is a diagonal matrix having only  $r$  nonzero entries, which makes the effective dimensions of these three matrices  $m \times r$ ,  $r \times r$ , and  $r \times n$ , respectively.  $U$  and  $V$  are two orthogonal matrices and  $S$  is a diagonal matrix, called the *singular matrix*. The diagonal entries  $(s_1, s_2, \dots, s_r)$  of  $S$  have the property that  $s_i > 0$  and  $s_1 \geq s_2 \geq \dots \geq s_r$ . The first  $r$  columns of  $U$  and  $V$  represent the orthogonal eigenvectors associated with the  $r$  nonzero eigenvalues of  $AA^T$  and  $A^TA$ , respectively. In other words, the  $r$  columns of  $U$  corresponding to the nonzero singular values span the *column space*, and the  $r$  columns of  $V$

span the *row space* of the matrix  $A$ .  $U$  and  $V$  are called the *left* and the *right* singular vectors, respectively.

SVD has an important property that makes it particularly interesting for our application. SVD provides the best *low-rank* linear approximation of the original matrix  $A$ . It is possible to retain only  $k \ll r$  singular values by discarding other entries. We term this reduced matrix  $S_k$ . Since the entries in  $S$  are sorted i.e.,  $s_1 \geq s_2 \geq \dots \geq s_r$ , the reduction process is performed by retaining the first  $k$  singular values. The matrices  $U$  and  $V$  are also reduced to produce matrices  $U_k$  and  $V_k$ , respectively. The matrix  $U_k$  is produced by removing  $(r - k)$  columns from the matrix  $U$  and matrix  $V_k$  is produced by removing  $(r - k)$  rows from the matrix  $V$ . When we multiply these three reduced matrices, we obtain a matrix  $A_k$ . The reconstructed matrix  $A_k = U_k \cdot S_k \cdot V_k^T$  is a *rank-k* matrix that is the closest approximation to the original matrix  $A$ . More specifically,  $A_k$  minimizes the *Frobenius norm*  $\|A - A_k\|_F$  over all rank- $k$  matrices. Researchers [1, 2] pointed out that the *low-rank* approximation of the original space is better than the original space itself due to the filtering out of the small singular values that introduce “noise” in the customer-product relationship.

The dimensionality reduction approach in SVD can be very useful for the collaborative filtering process. SVD produces a set of uncorrelated eigenvectors. Each customer and product is represented by its corresponding eigenvector. The process of dimensionality reduction may help customers who rated similar products (but not exactly the same products) to be mapped into the space spanned by the same eigenvectors. We now present an outline of the prediction generation algorithm using SVD (see [9] for details).

**Prediction generation using SVD.** Once the  $m \times n$  ratings matrix  $R$  is decomposed and reduced into three SVD component matrices with  $k$  features  $U_k$ ,  $S_k$ , and  $V_k$ , prediction can be generated from it by computing the cosine similarities (dot products) between  $m$  pseudo-customers  $U_k \cdot \sqrt{S_k}^T$  and  $n$  pseudo-products  $\sqrt{S_k} \cdot V_k^T$  [1]. In particular, the prediction score  $P_{i,j}$  for the  $i$ -th customer on the  $j$ -th product by adding the row average  $\bar{r}_i$  to the similarity. Formally,  $P_{i,j} = \bar{r}_i + U_k \cdot \sqrt{S_k}^T(i) \cdot \sqrt{S_k} \cdot V_k^T(j)$ . Once the SVD decomposition is done, the prediction generation process involves only a dot product computation, which takes  $\mathcal{O}(1)$  time, since  $k$  is a constant.

### 2.2 Challenges of Dimensionality Reduction

In a typical recommender system, the entire algorithm works in two separate steps. The first step is the *off-line* or *model-building* step and the second step is the *on-line* or the *execution* step. The user-user similarity computation and neighborhood formation [10] can be thought of as the off-line step of a CF system, whereas

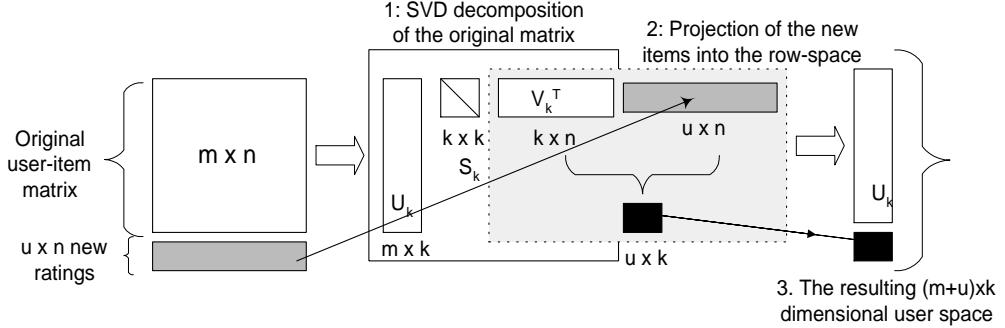


Figure 1: Schematic diagram of the SVD folding-in technique.

the actual prediction generation is the online step. Usually, the off-line component is very time-consuming and is computed relatively infrequently. For instance, an e-commerce site may compute the user-user similarity only once a day or even once a week. This works well if the ratings database is static and if the user behavior does not change significantly over a short period of time.

Researchers have demonstrated that the SVD-based algorithms can make the neighborhood formation process of CF systems highly scalable while producing better results in most of the cases [4, 9, 10]. Despite the good quality and excellent on-line performance SVD based algorithms suffer a serious drawback—the off-line SVD decomposition step is computationally very expensive. For an  $m \times n$  user-item matrix, the SVD decomposition requires a run-time of  $\mathcal{O}(m)^3$  [1, 2]. Our focus is to develop algorithms that ensure highly scalable overall performance. In order to achieve that goal, we must ensure that both the online and the off-line algorithms become more scalable. In the following section, we devise an algorithm that makes the off-line model building of SVD more scalable while achieving prediction quality comparable to the original SVD scheme.

### 3 Incremental SVD Algorithms

SVD has a property that allows the model to be incrementally computed. This method was used by the LSI researchers [1, 2] to handle dynamic databases, where new terms and documents may arrive once the model is built. It was shown that a projection of additional terms and documents can potentially provide a fairly good approximation of the model. We extend this idea to build a system where we first compute a suitably sized model and then use the projection method to build incrementally upon that. The resulting model is not a perfect SVD model as the space is not orthogonal, but the quality is expected to be good with potentially high performance gain.

**The Algorithm.** The projection technique is known as *folding-in* in SVD literature [1, 2]. To fold-in new users into the space of already reduced user-item ma-

trix  $A_k$ , we compute the coordinates for that vector in the basis  $U_k$ . Let the size of the new user vector  $N_u$  be  $t \times 1$ . The first step in folding-in is to compute a projection  $P$  that projects  $N_u$  onto the space. Such a projection  $P$  of  $N_u$  is computed as:

$$P = U_k \times U_k^T \times N_u. \quad (2)$$

This user set is then folded-in by appending the  $k$  dimensional vector  $U_k^T \cdot N_u$  as a new column of the  $k \times d$  matrix  $S_k \cdot V_k^T$ . Figure 1 shows a schematic diagram of the folding-in approach.

The folding-in technique allows us to devise a model-based approach for SVD-based prediction algorithms. Folding-in is based on the existing model ( $U_k$ ,  $S_k$ , and  $V_k$ ) and hence, new users or items do not affect existing user and items. In practice, it is possible to pre-compute the SVD decomposition using  $m$  existing users. For a user-item ratings matrix  $A$ , the three decomposed matrix  $U_k$ ,  $S_k$  and  $V_k$  are computed at first. As described in the previous section, these matrices can be used for prediction generation. However, when a new set of ratings is added to the database, it is not necessary to re-compute the low-dimensional model from the scratch. We can take advantage of the folding-in technique to build an incremental system that has the potential to be highly scalable.

## 4 Experimental Evaluation

This section describes our experimental verification of the incremental SVD algorithm. We first present our experimental platform—the data set, the evaluation metric, and the computational environment. Then we present our experimental procedure followed by the results and discussion.

### 4.1 Experimental platform

**Data set.** We used data from MovieLens ([www.movieLens.umn.edu](http://www.movieLens.umn.edu)), which is our web-based research recommender system that debuted in Fall 1997. We randomly selected enough users to obtain 100,000 ratings from the database (we only considered

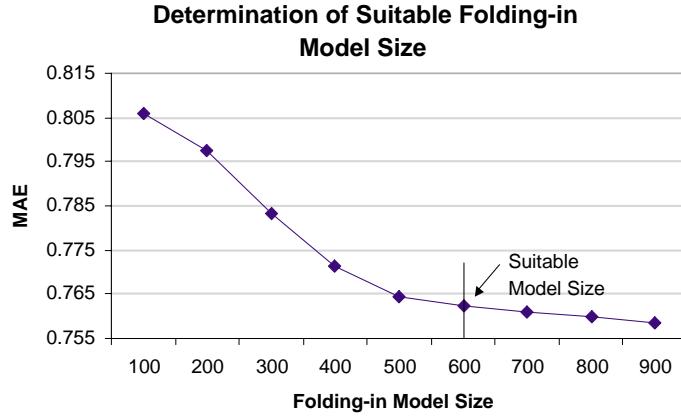


Figure 2: Determination of the threshold basis size for folding-in based SVD system

users that had rated 20 or more movies). The data set was converted into a user-movie matrix  $R$  that had 943 rows(users) and 1682 columns (movies). For our experiments, we divided the data set into a *training* and a *test* portion. We varied the training and test data ratio by using a parameter  $x$ , where  $x = 0.8$  means that 80% data was used for training the algorithm and 20% was used as test.

**Evaluation metric.** For our experiments, we use a widely popular statistical accuracy metric named *Mean Absolute Error (MAE)*, which is a measure of the deviation of recommendations from their true user-specified values [5, 11]. For each ratings-prediction pair  $\langle p_i, q_i \rangle$ , this metric treats the absolute error between them i.e.,  $|p_i - q_i|$  equally. The MAE is computed by first summing these absolute errors of the  $N$  corresponding ratings-prediction pairs and then computing the average. Formally,  $MAE = \frac{\sum_{i=1}^N |p_i - q_i|}{N}$ . The lower the MAE, the more accurately the recommendation engine predicts user ratings.

**Environment.** All our experiments were done using a combination of MATLAB and C, running on a Linux platform. The machine had 650 MHz Intel Pentium III CPU with 256 MB of RAM, and 512 KB of cache memory.

## 4.2 Experimental Procedure

For this experiment, we use the prediction generation algorithm using SVD described in [9], but instead of computing the SVD model (decomposition of matrix  $A$  into matrices  $U, S$ , and  $V$ ) for all users, we use a *threshold* size to build an initial model and then use the folding-in technique to incrementally compute the SVD model for additional users. Before performing the prediction experiments, we first determine the optimal values of our two experimental parameters—*i*) the

number of dimensions  $k$ , and *ii*) the threshold model size (basis size). We then perform the folding-in step and generate predictions using the incremental model. Finally, we investigate the performance implications of the folding-in technique. We used 10-fold cross validation by selecting random training and test data for all our experiments.

### 4.2.1 Values of experimental parameters

**Optimal value of  $k$ .** To determine the optimal value of the number of dimension  $k$ , we performed an experiment where we generated predictions using different dimensions each time. We plotted our results and from there obtained that 14 is the suitable value of  $k$ . For the rest our experiments we use  $k = 14$ .

**Optimal value of the basis size.** Our goal is to select a basis size that is small enough to produce fast model building yet large enough to produce good prediction quality. If we start with a very small basis size, the entire model computation can be very fast, but due to non-orthogonal spaces the prediction quality may not be good. On the other hand a large basis size will defeat the purpose of incremental model building. We determined the optimal basis size through experiments, where we first fix a basis size and compute the SVD model by projecting the rest of (*total – basis*) users using the folding-in technique. We start with a model size of 100 and go up to 900 with an increment of 100 at each step. Finally, we apply MAE to evaluate the prediction quality. We observe from Figure 2 that the quality of predictions improved as we increased the basis size. We further noticed that after the basis size crossed 600, the improvement in MAE values became relatively small. From this observation, we select 600 to be our threshold basis size.

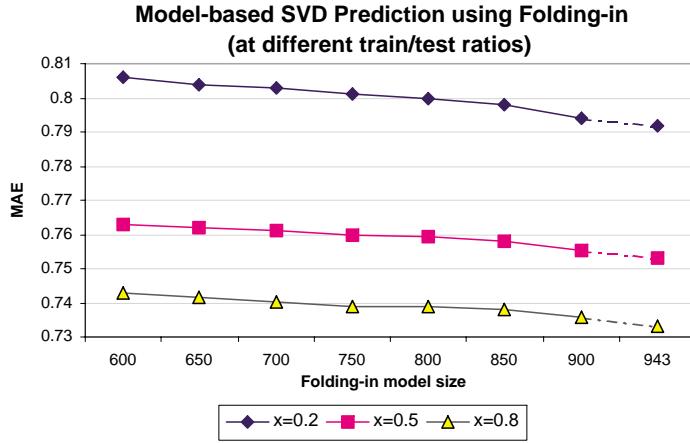


Figure 3: Prediction quality of folding-in based algorithms

#### 4.2.2 Results and Discussion

**Prediction quality experiments.** Figure 4(a) plots the prediction quality results with the incremental model building using folding-in technique. We report our results at three different training-test ratios—at  $x = 0.2, 0.5$ , and  $0.8$ . For each experiment, we start with a model size equal to the threshold basis size for folding-in. Then we use the projection method to fold-in the rest of ( $943 - \text{basis}$ ) users onto the SVD space. This process gives us an approximate SVD model for 943 users, where unlike the original SVD model the component matrices  $U, S$ , and  $V$  are not orthogonal. We then use the component matrices to generate prediction for a test set of ratings. We start with a model size of 600 and go up to 900 with an increment of 50.

The plots of Figure 3 show that even with a small basis size it is possible to obtain a good quality. The MAE value at  $x = 0.8$ , for example, is 0.733 for the full model size and 0.742 for a model size of 600 (only 1.22% quality drop!). Similar numbers can also be found at other  $x$  values. This suggests that the inexpensive projection technique provides good quality even with a small basis size.

**Performance implications** The observation from Figure 4 that the quality does not change dramatically with varying model size suggests that the SVD prediction generation system can be made more scalable by using the folding-in method. To investigate these scalability impacts, we record the run-time in seconds for each run and from there, we compute the throughput performance metric. The throughput plot, presented in Figure 4(b), shows the number of predictions generated per second at different basis sizes. From the plot corresponding to  $x = 0.8$  and the basis size of 600, we have a test case size of  $(100,000 * (1 - 0.8))$ . This means that our algorithm generates 20,000 ratings in 408.27 seconds, from there we obtain a throughput rate of 88.82 recommendations per second. Accordingly, at

a full model size of 943 the throughput becomes 48.9 (81.63% performance gain!). This difference is even more prominent at lower values of  $x$ , where the workload size is larger.

Overall, the folding-in technique shows the potential to be very useful in addressing the scalability challenge of SVD-based prediction generation systems. We have demonstrated that although folding-in results in slight quality loss due to the non-orthogonality of the resultant space, it shows substantial performance gain.

## 5 Conclusion and Future Work

SVD-based recommendation generation technique leads to very fast online performance, requiring just a few simple arithmetic operations for each recommendation but computing the SVD is very expensive. Use of incremental SVD algorithms such as folding-in [1] can significantly speed up the SVD computation cost while providing comparable prediction quality. In this paper, we have demonstrated that incremental SVD algorithms, based on folding-in, can help recommender systems achieve high scalability while providing good predictive accuracy.

The folding-in technique requires less time and storage space but can result in the loss of quality due to the non-orthogonality of the incremental SVD space. Researchers [1, 2] have pointed out techniques to incrementally update the space by retaining the orthogonality. This method is known as the *SVD-update* and requires more time and memory than the folding-in technique. Zha *et al.* [12] points out that the updating technique described in [1] is, in fact, inaccurate and they provide modified and very complex mathematical technique to implement the updating technique they claim to be more accurate. Implementation of this technique remains as a future work.

Future work is required to understand exactly why SVD works well for some recommender applications,

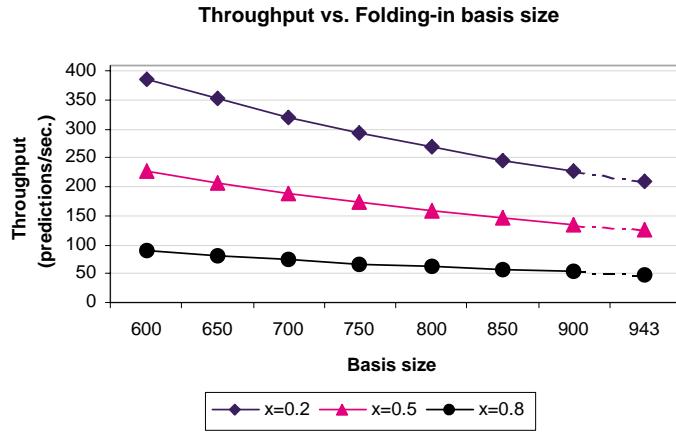


Figure 4: Throughput of the SVD folding-in algorithm

and less well for others. Also, there are many other ways in which SVD could be applied to recommender systems problems, including using SVD to create low-dimensional visualizations of the ratings space or using SVD to identify significant products that would help bootstrapping the recommender systems.

## 6 Acknowledgments

Funding for this research was provided in part by the National Science Foundation under grants IIS 9613960, IIS 9734442, and IIS 9978717 with additional funding by Net Perceptions Inc. This work was also supported by NSF CCR-9972519, by Army Research Office contract DA/DAAG55-98-1-0441, by the DOE ASCI program and by Army High Performance Computing Research Center contract number DAAH04-95-C-0008. Access to computing facilities was provided by AH-PCRC and Minnesota Supercomputer Institute. We also thank anonymous reviewers for their valuable comments.

## References

- [1] Berry, M. W., Dumais, S. T., and O'Brian, G. W. (1995). Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review*, 37(4).
- [2] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*. 41(6).
- [3] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM*. December.
- [4] Gupta, D., and Goldberg, K. (1999). Jester 2.0: A Linear Time Collaborative Filtering Algorithm Applied to Jokes. In *Proc. of the ACM SIGIR '99*.
- [5] Herlocker, J. L., Konstan, J. A., Borchers, A., and Riedl, J. (1999). An Algorithmic Framework for Performing Collaborative Filtering. In *Proc. of ACM SIGIR'99*.
- [6] Hill, W., Stead, L., Rosenstein, M., and Furnas, G. (1995). Recommending and Evaluating Choices in a Virtual Community of Use. In *Proc. of CHI '95*.
- [7] Resnick, P. and Varian, H. R. (1997). Recommender Systems. *Communications of the ACM*. 40(3), pp. 56-58.
- [8] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proc. of CSCW '94*, Chapel Hill, NC.
- [9] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. (2000). Application of Dimensionality Reduction in Recommender System—A Case Study. In *ACM WebKDD'00 (Web-mining for E-Commerce Workshop)*.
- [10] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. (2000). Analysis of Recommendation Algorithms for E-Commerce. In *Proc. of the ACM EC'00 Conference*. Minneapolis, MN, pp. 158-167.
- [11] Shardanand, U., and Maes, P. (1995). Social Information Filtering: Algorithms for Automating 'Word of Mouth'. In *Proc. of CHI '95*. Denver, CO.
- [12] Zha, H., and Zhang, Z. (1999). On matrices with Low-Rank-Plus-Shift Structure: Partial SVD and Latent Semantic Indexing. *SIAM Journal of Matrix Analysis and Applications*, vol. 21.

# Item-Based Top- $N$ Recommendation Algorithms

MUKUND DESHPANDE and GEORGE KARYPIS  
University of Minnesota

---

The explosive growth of the world-wide-web and the emergence of e-commerce has led to the development of *recommender systems*—a personalized information filtering technology used to identify a set of items that will be of interest to a certain user. User-based collaborative filtering is the most successful technology for building recommender systems to date and is extensively used in many commercial recommender systems. Unfortunately, the computational complexity of these methods grows linearly with the number of customers, which in typical commercial applications can be several millions. To address these scalability concerns model-based recommendation techniques have been developed. These techniques analyze the user-item matrix to discover relations between the different items and use these relations to compute the list of recommendations.

In this article, we present one such class of model-based recommendation algorithms that first determines the similarities between the various items and then uses them to identify the set of items to be recommended. The key steps in this class of algorithms are (i) the method used to compute the similarity between the items, and (ii) the method used to combine these similarities in order to compute the similarity between a *basket* of items and a candidate recommender item. Our experimental evaluation on eight real datasets shows that these *item-based* algorithms are up to two orders of magnitude faster than the traditional user-neighborhood based recommender systems and provide recommendations with comparable or better quality.

Categories and Subject Descriptors: H.2.8 [Database Management]: Database Applications—*data mining*; H.3.3 [Information Storage and Retrieval]: Search and Retrieval—*information filtering*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: world wide web, e-commerce, predicting user behavior.

---

This work was supported in part by National Science Foundation (NSF) grants EIA-9986042, ACI-9982274, and ACI-0133464; NASA NCC 21231, the Digital Technology Center at the University of Minnesota; and by the Army High-Performance Computing Research Center (AHPCRC) under the auspices of the Department of the Army, Army Research Laboratory (ARL) under Cooperative Agreement number DAAD19-01-2-0014.

The content of this article does not necessarily reflect the position or the policy of the government and no official endorsement should be inferred.

Contact author: G. Karypis, Dept. of Computer Science & Engineering, 4-192 EE/CS Building, 200 Union Street SE, Minneapolis, MN 55455; email: karypis@cs.umn.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.

© 2004 ACM 1046-8188/04/0100-0143 \$5.00

## 1. INTRODUCTION

The explosive growth of the world-wide-web and the emergence of e-commerce has led to the development of *recommender systems* [Resnick and Varian 1997]. Recommender systems are personalized information filtering technology used to either predict whether a particular user will like a particular item (*prediction problem*) or to identify a set of  $N$  items that will be of interest to a certain user (*top- $N$  recommendation problem*). In recent years, recommender systems have been used in a number of different applications [Shardanand and Maes 1995; Hill et al. 1995; Konstan et al. 1997; Terveen et al. 1997; Schafer et al. 1999; Kitts et al. 2000; Mobasher et al. 2000; Beeferman and Berger 2000] such as recommending products a customer will most likely buy; movies, TV programs, or music a user will find enjoyable; identifying web pages that will be of interest; or even suggesting alternate ways of searching for information. An excellent survey of different recommender systems for various applications can be found in Schafer et al. [1999] and Resnick and Varian [1997].

Over the years, various approaches for building recommender systems have been developed that utilize either demographic, content, or historical information [Hill et al. 1995; Balabanovic and Shoham 1997; Basu et al. 1998; Shardanand and Maes 1995; Terveen et al. 1997; Konstan et al. 1997]. Among them, collaborative filtering (CF), which relies on historical information, is probably the most successful and widely used technique for building recommender systems [Resnick et al. 1994; Konstan et al. 1997]. The term, *collaborative filtering* was first coined in Goldberg et al. [1992], where it was used to describe an e-mail filtering system called Tapestry, which was designed to filter e-mails received from mailing lists and newsgroup postings. In this system, each user could write a comment (annotation) about each e-mail message and share these annotations with a group of users. A user could then filter these e-mail messages by writing queries on these annotations. Though Tapestry allowed an individual user to benefit from annotations made by other users, the system required an individual user to write complicated queries. The first system to generate automated recommendations was the GroupLens system [Resnick et al. 1994; Konstan et al. 1997], which provided users with personalized recommendations on Usenet postings. The recommendations for each individual were obtained by identifying a neighborhood of similar users and recommending the articles that this group of users found useful.

Two approaches have been developed for building CF-based *top- $N$*  recommender systems. The first approach, referred to as *user-based* [Shardanand and Maes 1995; Konstan et al. 1997; Breese et al. 1998; Resnick et al. 1994; Herlocker et al. 1999; Sarwar et al. 2000], relies on the fact that each person belongs in a larger group of similarly behaving individuals. As a result, items (e.g., products, movies, books, etc.) frequently purchased/liked by the various members of the group can be used to form a basis for recommended items. The second approach, known as *model-based* [Shardanand and Maes 1995; Billsus and Pazzani 1998; Breese et al. 1998; Aggarwal et al. 1999; Kitts et al. 2000], analyzes historical information to identify relations between different items such that the purchase of an item (or a set of items) often leads to the purchase

of another item (or a set of items), and then use these relations to determine the recommended items. Model-based schemes, by using precomputed models, produce recommendations very quickly but tend to require a significant amount of time to build these models. Furthermore, these recommendations are generally of lower-quality than those produced by user-based schemes. In contrast, user-based schemes tend to produce systems that lead to higher-quality recommendations but suffer serious scalability problems as the complexity of computing each recommendation grows linearly with the number of users and items.

The focus of this article is on a particular class of model-based *top-N* recommendation algorithms that build the recommendation model by analyzing the similarities between the various items and then use these similar items to identify the set of items to be recommended. These algorithms, referred to in this article as *item-based top-N recommendation algorithms*, have been used in various forms since the early days of CF-based recommender systems [Shardanand and Maes 1995; Kitts et al. 2000] and were shown to be computationally scalable (both in terms of model construction and model application) but tended to produce lower-quality recommendations when compared to user-based schemes.

The contributions of this article are two-fold. First, we present a detailed study of the two key steps that affect the performance of item-based *top-N* recommendation algorithms, which are (i) the method used to compute the similarity between the items and (ii) the method used to combine these similarities in order to compute the similarity between a *basket* of items and a candidate recommender item. For the first step, we study two different methods of computing the item-to-item similarity. One models the items as vectors in the user space, and uses the *cosine* function to measure the similarity, whereas the other computes the item-to-item similarities using a technique based on the conditional probability between two items. This conditional probability technique is extended so that it can differentiate between users with varying amounts of historical information as well as between frequently and infrequently purchased items. For the second step, we present a method for combining these similarities that accounts for item-neighborhoods of different density that can incorrectly bias the overall recommendation.

The second contribution is the extension of these item-based schemes to higher-order models, which obtain the final recommendations by exploiting relations between sets of items. We present a class of *interpolated higher-order item-based top-N recommendation* algorithms that construct a recommendation model by first determining the various itemset-item similarities and then combining them to determine the similarity between a user's *basket* and a candidate recommender item.

We present a detailed experimental evaluation of these algorithms and study the performance implications of the various parameters on two classes of datasets. The first class consists of eight real datasets arising in various applications, whereas the second class consists of 36 datasets that were synthetically generated by the widely-used synthetic transaction dataset generator provided by the IBM Quest group [Agrawal and Srikant 1994]. Our experiments show that the item-based algorithm when combined with the conditional probability-based similarity method produce higher-quality recommendations

than the user-based scheme on both real and synthetic datasets. Moreover, the higher-order schemes lead to additional improvements when the density of the datasets increases and when the users have many items in common. Furthermore, our computational complexity evaluation shows that the item-to-item based algorithms are up to two orders of magnitude faster than the traditional user-based algorithms. Some of the results in this paper were previously presented in Karypis [2001].

The paper is organized as follows: Section 2 provides the definitions and notations that will be used throughout the paper. Section 3 presents a brief survey of the related research on collaborative filtering-based recommender algorithms. Sections 4 and 5 describe the various phases and algorithms used in our first- and higher-order item-based *top-N* recommendation system. Section 6 provides an experimental evaluation of the various parameters of the proposed algorithms, and compares the proposed algorithms against user-based ones. Finally, Section 7 provides some concluding remarks.

## 2. DEFINITIONS AND NOTATIONS

Throughout the article, we will use the symbols  $n$  and  $m$  to denote the number of distinct users and the number of distinct items in a particular dataset, respectively. We will use the symbol  $N$  to denote the number of recommendations that needs to be computed for a particular user. In presenting the various algorithms we will assume that the underlying application domain is that of commercial retailing and we will use the terms *customers* and *products* as synonyms to users and items, respectively. We will use the term *dataset* to denote the set of transactions about the items that have been purchased by the various users. We will represent each dataset by an  $n \times m$  binary matrix  $R$  that will be referred to as the *user-item matrix*, such that  $R_{i,j}$  is one if the  $i$ th customer has purchased the  $j$ th item, and zero otherwise. We will refer to the user for which we want to compute the *top-N* recommendations as the *active user*, and to the set of items that the user has already purchased as the user's *basket*. Finally, the *top-N* recommendation problem is formally defined as follows:

*Definition 2.1 (top-N Recommendation Problem).* Given a user-item matrix  $R$  and a set of items  $U$  that have been purchased by a user, identify an ordered set of items  $X$  such that  $|X| \leq N$  and  $X \cap U = \emptyset$ .

## 3. RELATED RESEARCH

User-based collaborative filtering is the most successful technology for building recommender systems to date and is extensively used in many commercial recommender systems. In general, user-based systems compute the *top-N* recommended items for a particular user by following a three-step approach [Shardanand and Maes 1995; Konstan et al. 1997; Sarwar et al. 2000]. In the first step, they identify the  $k$  users in the database that are the most similar to the active user. During the second step, they compute the union of the items purchased by these users and associate a weight with each item based on its importance in the set. In the third and final step, from this union they select and recommend the  $N$  items that have the highest weight and have not

already been purchased by the active user. Within this three-step framework, the method used to determine the  $k$  most similar users and the scheme used to determine the importance of the different items play the most critical role in the overall performance of the algorithm. Commonly, the similarity between the users is computed by treating them as vectors in the item-space and measuring their similarity via the cosine or correlation coefficient functions [Breese et al. 1998; Sarwar et al. 2000], whereas the importance of each item is determined by how frequently it was purchased by the  $k$  most similar users. However, alternate approaches for both of these steps have been explored and shown to lead to somewhat better results. A detailed survey of different user-based algorithms and a comparison of their performance can be found in Breese et al. [1998], Herlocker et al. [1999], and Sarwar et al. [2000].

Despite the popularity of user-based recommender systems, they have a number of limitations related to scalability and real-time performance. The computational complexity of these methods grows linearly with the number of customers, which in typical commercial applications can grow to be several millions. Furthermore, even though the user-item matrix is sparse, the user-to-user similarity matrix is quite dense. This is because even a few frequently purchased items can lead to dense user-to-user similarities. Moreover, real-time  $\text{top-}N$  recommendations based on the current basket of items, utilized by many e-commerce sites, cannot take advantage of pre-computed user-to-user similarities. Finally, even though the throughput of user-based recommendation algorithms can be increased by increasing the number of servers running the recommendation algorithm, they cannot decrease the latency of each  $\text{top-}N$  recommendation, which is critical for near real-time performance. One way of reducing the complexity of the nearest-neighbor computations is to cluster the users and then to either limit the nearest-neighbor search among the users that belong to the nearest cluster, or use the cluster centroids to derive the recommendations [Ungar and Foster 1998; Mobasher et al. 2000]. These approaches, though they can significantly speed up the recommendation algorithm, tend to decrease the quality of the recommendations.

To address the scalability concerns of user-based recommendation algorithms a variety of model-based recommendation techniques were developed. Billsus and Pazzani [1998] developed a model-based recommender system by treating the  $\text{top-}N$  recommendation problem as a classification problem, in which the goal was to classify the items purchased by an individual user into two classes: like and dislike. A classification model based on neural networks was built for each individual user where the items purchased by the user were thought of as the examples and the users as the attributes. A singular value decomposition of the user-item matrix reduced the dimensionality of the problem. The prediction on an item was computed by constructing an example for that item and feeding it to the classifier. The authors reported considerable improvements over the traditional user-based algorithms. Though this approach is quite powerful it requires building and maintaining a neural network model for each individual user in the database, which is not scalable to large databases. Breese et al. [1998] presented two model-based algorithms for computing both predictions and  $\text{top-}N$  recommendations. The first algorithm follows a probabilistic

approach in which the users are clustered and the conditional probability distribution of different items in the cluster is estimated. The probability that the active user belongs to a particular cluster given the basket of items is then estimated from the clustering solution and the probability distribution of items in the cluster. The clustering solution for this technique is computed using the expectation maximization (EM) principle. The second algorithm is based on Bayesian network models where each item in the database is modeled as a node having states corresponding to the rating of that item. The learning problem consists of building a network on these nodes such that each node has a set of parent nodes that are the best predictors for the child's rating. They presented a detailed comparison of these two model-based approaches with the user-based approach and showed that Bayesian networks model outperformed the clustering model as well as the user-based scheme. Heckerman et al. [2000] proposed a recommendation algorithm based on dependency networks instead of Bayesian networks. Though the accuracy of dependency networks is inferior to Bayesian networks they are more efficient to learn and have smaller memory requirements. Aggarwal et al. [1999] presented a graph-based recommendation algorithm where the users are represented as the nodes in a graph and the edges between the nodes indicate the degree of similarity between the users. The recommendations for a user are computed by traversing nearby nodes in this graph. The graph representation of the model allows it to capture transitive relations which cannot be captured by nearest neighbor algorithms and the authors reported better performance than the user-based schemes.

A number of different model-based approaches have been developed that use item-to-item similarities as well as association rules. Shardanand and Maes [1995] developed an item-based prediction algorithm within the context of the Ringo music recommendation system, referred to as *artist-artist*, that determines whether or not a user will like a particular artist by computing its similarity to the artists that the user has liked/disliked in the past. This similarity was computed using the Pearson correlation function. Sarwar et al. [2001] further studied this paradigm for computing predictions and they evaluated various methods for computing the similarity as well as approaches to limit the set of item-to-item similarities that need to be considered. The authors reported considerable improvements in performance over the user-based algorithm. Mobasher et al. [2000] presented an algorithm for recommending additional webpages to be visited by a user based on association rules. In this approach, the historical information about users and their web-access patterns were mined using a frequent itemset discovery algorithm and were used to generate a set of high confidence association rules. The recommendations were computed as the union of the consequent of the rules that were supported by the pages visited by the user. Lin et al. [2000] used a similar approach but they developed an algorithm that is guaranteed to find association rules for all the items in the database. Finally, within the context of using association rules to derive *top-N* recommendations, Demiriz [2001] studied the problem of how to weight the different rules that are supported by the active user. He presented a method that computes the similarity between a rule and the active user's basket as the product of the confidence of the rule and the Euclidean distance

between items in the antecedent of the association rule and the items in the user's basket. He compared this approach both with the item-based scheme described in Section 4 (based on our preliminary work presented in Karypis [2001]) and the dependency network-based algorithm [Heckerman et al. 2000]. His experiments showed that the proposed association rule-based scheme is superior to dependency networks but inferior to the item-based schemes.

#### 4. ITEM-BASED $TOP-N$ RECOMMENDATION ALGORITHMS

In this section, we study a class of model-based  $top-N$  recommendation algorithms that use item-to-item similarities to compute the relations between the different items. The primary motivation behind these algorithms is the fact that a customer is more likely to purchase items that are similar to the items that he/she has already purchased in the past; thus, by analyzing historical purchasing information (as represented in the user-item matrix) we can automatically identify these sets of similar items and use them to form the  $top-N$  recommendations. These algorithms are similar in spirit to previously developed item-based schemes [Shardanand and Maes 1995; Kitts et al. 2000] but differ in a number of key aspects related to how the similarity between the different items is computed and how these similarities are combined to derive the final recommendations.

At a high-level, these algorithms consist of two distinct components. The first component builds a model that captures the relations between the different items, whereas the second component applies this precomputed model to derive the  $top-N$  recommendations for an active user. The details on these components are presented in the remainder of this section.

##### 4.1 Building the Model

The model used by the item-based  $top-N$  recommendation algorithm is constructed using the algorithm shown in Algorithm 4.1. The input to this algorithm is the  $n \times m$  user-item matrix  $R$  and a parameter  $k$  that specifies the number of item-to-item similarities that will be stored for each item. The output is the model itself, which is represented by an  $m \times m$  matrix  $\mathcal{M}$  such that the  $j$ th column stores the  $k$  most similar items to item  $j$ . In particular, if  $\mathcal{M}_{i,j} > 0$ , then the  $i$ th item is among the  $k$  most similar items of  $j$  and the value of  $\mathcal{M}_{i,j}$  indicates the degree of similarity between items  $j$  and  $i$ .

**Algorithm 4.1:** BUILDMODEL ( $R, k$ )

```

for  $j \rightarrow 1$  to  $m$ 
  do {
    for  $i \rightarrow 1$  to  $m$ 
      do {
        if  $i \neq j$ 
          then  $\mathcal{M}_{i,j} \rightarrow \text{sim}(R_{*,j}, R_{*,i})$ 
          else  $\mathcal{M}_{i,j} \rightarrow 0$ 
        for  $i \rightarrow 1$  to  $m$ 
          do {
            if  $\mathcal{M}_{i,j} \neq$  among the  $k$  largest values in  $\mathcal{M}_{*,j}$ 
              then  $\mathcal{M}_{i,j} \rightarrow 0$ 
      return ( $\mathcal{M}$ )
    }
  }
}

```

(1)

(2)

The parameterization of  $\mathcal{M}$  on  $k$  was motivated due to performance considerations and its choice represents a performance-quality trade-off. By using a small value of  $k$ , we can ensure that  $\mathcal{M}$  is very sparse and thus can be stored in main memory even in collaborative filtering environments and applications in which  $m$  is very large. However, if  $k$  is too small, then the resulting model will contain limited information from which to build the recommendations, and thus it can potentially lead to lower quality. Fortunately, as our experimental evaluation will illustrate (Section 6.2.1), reasonably small values of  $k$  ( $10 \leq k \leq 30$ ) lead to good results and higher values lead to either a very small or no improvement.

The actual algorithm for constructing  $\mathcal{M}$  is quite simple. For each item  $j$ , the algorithm computes the similarity between  $j$  and the other items and stores the results in the  $j$ th column of  $\mathcal{M}$  (line 1). Once these similarities have been computed, it then proceeds to zero-out all the entries in the  $j$ th column of  $\mathcal{M}$  that contain smaller values than the  $k$  largest similarity values in that column. The resulting matrix  $\mathcal{M}$  that contains at most  $k$  nonzero entries per column becomes the final model of the item-based algorithm. Note that by construction (line 2) the algorithm ensures that a particular item will not contain itself as one of its  $k$  most similar items. This is done to ensure that an item does not contribute towards recommending itself. Such recommendations are of little value because we require the recommended items to be different from the items in the active user's basket.

**4.1.1 Measuring the Similarity between Items.** The properties of the model  $\mathcal{M}$  and consequently the effectiveness of the overall recommendation algorithm depend on the method used to compute the similarity between the various items (line 1 in Algorithm 4.1). In general, the similarity between two items  $i$  and  $j$  should be high if there are lot of customers that have purchased both of them, and it should be low if there are few such customers. There are also two somewhat less obvious aspects that we need to consider. The first has to do with whether or not we should be discriminating between customers that purchase few items and customers that purchase many items. For example, consider two customers  $C_1$  and  $C_2$ , both of whom have purchased items  $i$  and  $j$ , but  $C_1$  has purchased 5 additional items whereas  $C_2$  has purchased 50 additional items. Should the fact that both of them purchased  $i$  and  $j$  contribute equally while determining the similarity between this pair of items? There may be cases in which the copurchasing information derived from customers that have bought fewer items is a more reliable indicator for the similarity of two copurchased items than the information derived from customers that tend to buy a large number of items. This is because the first group represents consumers that are focused in certain product areas. As our experiments in Section 6.2.1 will show, this is often the case, and being able to take it into account improves the overall *top-N* recommendation performance.

The second aspect that we need to consider has to do with whether or not the similarity between a pair of items should be symmetric (i.e.,  $\text{sim}(i, j) = \text{sim}(j, i)$ ) or not (i.e.,  $\text{sim}(i, j) \neq \text{sim}(j, i)$ ). This question usually arises when we need to compute the similarity between pairs of items that are purchased

at substantially different frequencies. For example, consider two items  $i$  and  $j$  such that  $i$  has been purchased significantly more frequently than  $j$ . Due to that frequency difference, the number of times that  $i$  and  $j$  are purchased together will be much smaller than the number of times that  $i$  is purchased alone. What should the similarity between  $i$  and  $j$  be? From  $i$ 's point of view, its similarity to  $j$  is low, because only a small fraction of its occurrences will co-occur with  $j$ . However, from  $j$ 's point of view, its similarity to  $i$  may be high, because a large fraction of its occurrences may co-occur with  $i$ . Thus, if we use an asymmetric similarity function, we will have that  $\text{sim}(i, j) < \text{sim}(j, i)$ . However, if we use a symmetric function the similarity between  $j$  and  $i$  (from  $j$ 's point of view) will generally end up being smaller than it would be in the asymmetric case, as it would have to account for  $i$ 's higher frequency. Each one of these two approaches has its advantages. A symmetric similarity function will tend to eliminate recommendations of very frequent items (that to a large extent are obvious), as these items will tend to be recommended only if other frequently purchased items are in the current basket. However, in datasets that have no items that are frequently purchased by the majority of the customers, a symmetric similarity function will unnecessarily penalize the recommendation of items whose frequency is relatively higher than the items that have been currently purchased by the active user.

In this study, we use two different similarity functions that are derived from the vector-space model and probabilistic methods, respectively. The key difference between them is that the first leads to similarities that are symmetric, whereas the second leads to similarities that are asymmetric. Furthermore, we have modified both similarity functions so that they can weight the customers differently based on how many products they have purchased. The details of these functions and their modifications are provided in the rest of this section.

**4.1.1.1 Cosine-Based Similarity.** One way of computing the similarity between two items is to treat each item as a vector in the space of customers and use the *cosine* between these vectors as a measure of similarity. Formally, if  $R$  is the  $n \times m$  user-item matrix, then the similarity between two items  $i$  and  $j$  is defined as the cosine of the  $n$  dimensional vectors corresponding to the  $i$ th and  $j$ th column of matrix  $R$ . The cosine between these vectors is given by

$$\text{sim}(i, j) = \cos(\vec{R}_{*,i}, \vec{R}_{*,j}) = \frac{\vec{R}_{*,j} \cdot \vec{R}_{*,i}}{\|\vec{R}_{*,i}\|_2 \|\vec{R}_{*,j}\|_2}, \quad (1)$$

where ' $\cdot$ ' denotes the vector dot-product operation. Note that since the cosine function measures the angle between the two vectors it is a symmetric similarity function. As a result, frequently purchased items will tend to be similar to other frequently purchased items and not to infrequently purchased items, and vice versa.

In its simplest form, the rows of  $R$  can correspond to the original binary purchase information, in which case, the cosine similarity function treats customers that purchase a small and a large number of items equally. However, each one of the rows can be scaled so that the resulting cosine-based similarity function will differentiate between these sets of customers. This can be done by

scaling each row to be of unit length (or any other norm). The effect of this scaling is that customers that have purchased fewer items will contribute a higher weight to the dot-product in Eq. (1) than customers that have purchased more items.

**4.1.1.2 Conditional Probability-Based Similarity.** An alternate way of computing the similarity between each pair of items  $i$  and  $j$  is to use a measure that is based on the conditional probability of purchasing one of the items given that the other has already been purchased. In particular, the conditional probability of purchasing  $j$  given that  $i$  has already been purchased  $P(j|i)$  is nothing more than the number of customers that purchase both items  $i$  and  $j$  divided by the total number of customers that purchased  $i$ , that is,

$$P(j|i) = \frac{\text{Freq}(ij)}{\text{Freq}(i)},$$

where  $\text{Freq}(X)$  is the number of customers that have purchased the items in the set  $X$ . Note that, in general,  $P(j|i) \neq P(i|j)$  and using this as a measure of similarity leads to asymmetric relations.

As discussed earlier, one of the limitations of using an asymmetric similarity function is that each item  $i$  will tend to have high conditional probabilities with items that are being purchased frequently. This problem has been recognized by researchers in information retrieval and recommender systems [Salton 1989; Breese et al. 1998; Kitts et al. 2000; Chan 1999]. The problem can be corrected by dividing  $P(j|i)$  with a quantity that depends on the occurrence frequency of item  $j$ . Two different methods have been proposed for achieving this. The first one, inspired from the inverse-document frequency scaling performed in information retrieval systems, multiplies  $P(j|i)$  by  $-\log_2(P(j))$  [Salton 1989], whereas the other one divides  $P(j|i)$  by  $P(j)$  [Kitts et al. 2000]. Note that this latter method leads to a symmetric similarity function. Our experiments have shown that this scaling greatly affects the performance of the recommender system and that the *optimal* scaling degree is problem dependent. For these reasons we use the following formula to compute the similarity between two items:

$$\text{sim}(i, j) = \frac{\text{Freq}(ij)}{\text{Freq}(i) \times (\text{Freq}(j))^{\alpha}}, \quad (2)$$

where  $\alpha$  is a parameter that takes a value between 0 and 1. Note that, when  $\alpha = 0$ , Eq. (2) becomes identical to  $P(j|i)$ , whereas if  $\alpha = 1$ , it becomes similar (up to a scaling factor) to the formulation in which  $P(j|i)$  is divided by  $P(j)$ .

The similarity function as defined in Eq. (2) does not discriminate between customers that have purchased different number of items. To achieve this discrimination and give higher weight to the customers that have purchased fewer items, we have extended the similarity measure of Eq. (2) in the following way: First we normalize each row of matrix  $R$  to be of unit length, and then we define the similarity between items  $i$  and  $j$  as:

$$\text{sim}(i, j) = \frac{\sum_{q: R_{q,j} > 0} R_{q,j}}{\text{Freq}(i) \times (\text{Freq}(j))^{\alpha}}. \quad (3)$$

The only difference between Eq. (3) and Eq. (2) is that instead of using the co-occurrence frequency we use the sum of the corresponding nonzero entries of the  $j$ th column in the user-item matrix. Since the rows are normalized to be of unit length, customers that have purchased more items will tend to contribute less to the overall similarity. This gives emphasis to the purchasing decisions of the customers that have bought fewer items.

#### 4.2 Applying the Model

The algorithm for applying the item-based model is shown in Algorithm 4.2. The input to this algorithm is the model  $\mathcal{M}$ , an  $m \times 1$  vector  $U$  that stores the items that have already been purchased by the active user, and the number of items to be recommended ( $N$ ). The active user's information in vector  $U$  is encoded by setting  $U_i = 1$  if the user has purchased the  $i$ th item and zero otherwise. The output of the algorithm is an  $m \times 1$  vector  $x$  whose nonzero entries correspond to the *top- $N$*  items that were recommended. The weight of these nonzero entries represent a measure of the *recommendation strength* and the various recommendations can be ordered in non-increasing recommendation strength weight. In most cases  $x$  will have exactly  $N$  nonzero entries; however, the actual number of recommendations can be less than  $N$  as it depends on the value of  $k$  used to build  $\mathcal{M}$  and the number of items that have already been purchased by the active user.

**Algorithm 4.2:** APPLYMODEL ( $\mathcal{M}, U, N$ )

```

 $x \leftarrow \mathcal{M} \cdot U$                                      (1)
for  $j \leftarrow 1$  to  $m$                                 (2)
  do  $\begin{cases} \text{if } U_j \neq 0 \\ \quad \text{then } x_j \leftarrow 0 \end{cases}$ 
for  $j \leftarrow 1$  to  $m$                                 (3)
  do  $\begin{cases} \text{if } x_j \neq \text{among the } N \text{ largest values in } x \\ \quad \text{then } x_j \leftarrow 0 \end{cases}$ 
return ( $x$ )

```

The vector  $x$  is computed in three steps. First, the vector  $x$  is computed by multiplying  $\mathcal{M}$  with  $U$  (line 1). Note that the nonzero entries of  $x$  correspond to the union of the  $k$  most similar items for each item that has already been purchased by the active user, and that the weight of these entries is nothing more than the sum of these similarities. Second, the entries of  $x$  that correspond to items that have already been purchased by the active user are set to zero (loop at line 2). Finally, in the third step, the algorithm sets to zero all the entries of  $x$  that have a value smaller than the  $N$  largest values of  $x$  (loop at line 3).

One potential drawback with Algorithm 4.2 is that the raw similarity between each item  $j$  and its  $k$  most similar items may be significantly different. That is, the item neighborhoods are of different density. This is especially true for items that are purchased somewhat infrequently, since a moderate overlap with other infrequently purchased items can lead to relatively high similarity values. Consequently, these items can exert strong influence in the selection of the *top- $N$*  items, sometimes leading to wrong recommendations. For this

reason, instead of using the actual similarities computed by the various methods described in Section 4.1.1, for each item  $j$  we first normalize the similarities so that they add-up to one. That is,  $\|\mathcal{M}_{*,j}\| = 1$ , for  $j = 1, \dots, m$ . As the experiments presented in Section 6 show, this always improves the  $top\text{-}N$  recommendation quality.

#### 4.3 Computational Complexity

The computational complexity of the item-based  $top\text{-}N$  recommendation algorithm depends on the amount of time required to build the model  $\mathcal{M}$  (i.e., for each item identify the other  $k$  most similar items), and the amount required to compute the recommendation using this model.

During the model building phase we need to compute the similarity between each item  $j$  and the other items in  $R$  and then select the  $k$  most similar items. The upper bound on the complexity of this step is  $O(m^2n)$  as we need to compute  $m(m - 1)$  similarities, each potentially requiring  $n$  operations. However, the actual complexity is significantly smaller because the resulting item-to-item similarity matrix is extremely sparse. In our datasets, the item-to-item similarity matrix was generally more than 99% sparse. The reason for these sparsity levels is that each customer purchases a relatively small number of items and the items they purchase tend to be clustered. Consequently, by using sparse data structures to store  $R$  and only computing the similarities between pairs of items that are purchased by at least one customer we can substantially reduce the computational complexity.

Finally, the time required to compute the  $top\text{-}N$  recommendations for an active user that has purchased  $q$  items is given by  $O(kq)$  because we need to access the  $k$  most similar items for each one of the items that the user has already purchased and identify the overall  $N$  most similar items.

### 5. HIGHER-ORDER ITEM-BASED $TOP\text{-}N$ RECOMMENDATION ALGORITHMS

Our discussion so far has focused on item-based  $top\text{-}N$  recommendation algorithms in which the recommendations were computed by taking into account relations between pairs of items, that is, for each item in the active user's basket, similar items were determined and these similar items were aggregated to obtain the desired  $top\text{-}N$  recommendations. These schemes effectively ignore the presence of other items in the active user's basket while computing the  $k$  most similar items for each item. Even though this allows such schemes to be computationally efficient, they can potentially lead to suboptimal recommendations when the joint distribution of a set of items is different from the distributions of the individual items in the set.

To solve this problem, we developed item-based  $top\text{-}N$  recommendation schemes that use all combinations of items (i.e., *itemsets*) up to a particular size  $l$  when determining the set of items to be recommended to a user. In this approach, during the model building phase, instead of only determining the  $k$  most similar items for each individual item, we do so for all possible itemsets up to a particular size  $l$ . During the model application time, we compute the  $top\text{-}N$  recommendations by combining these sets of  $k$  item-neighborhoods not

just for individual items, but for all itemsets up to size  $l$  that are present in the active user's basket.

We will refer to the parameter  $l$  as the *order* of the item-based model, and we will refer to this class of item-based *top-N* recommendation algorithms as the *interpolated higher-order models*. When  $l = 1$ , the above scheme becomes identical to the one described in Section 4 and for this reason we will sometimes refer to it as the first-order model. The name, *interpolated*, was motivated by the interpolated Markov models used in DNA sequence analysis [Delcher et al. 1998] and is used to indicate that the final predictions are computed by combining models that use itemsets of different size (i.e., the final solution is an *interpolation* of predictions computed by models that use one, two, ..., up to  $l$  itemsets).

The remainder of this section describes these higher-order item-based *top-N* recommendation algorithms in detail and discusses various issues associated with their efficient implementation.

### 5.1 Building the Model

During the model building phase we use the algorithm shown in Algorithm 5.1 to compute  $l$  different model matrices  $\mathcal{M}^1, \mathcal{M}^2, \dots, \mathcal{M}^l$  of size  $m \times m, m \times m^2, \dots, m \times m^l$ , respectively. For a particular value of  $r$ ,  $\mathcal{M}^r$  is constructed by generating all possible combinations of  $r$  items  $\{q_1, q_2, \dots, q_r\}$  (loop at line 1), computing the similarity between these sets and all the other  $m$  items in the dataset (loop at line 2), and among them retaining only the  $k$  largest similarities in the corresponding columns of  $\mathcal{M}^r$  (loop at line 3).

**Algorithm 5.1:** BUILDHIGHERORDERMODEL ( $R, l, k$ )

```

for  $r \leftarrow 1$  to  $l$ 
  do  $\left\{ \begin{array}{l} \textbf{for } j \leftarrow 1 \textbf{ to } m^r \\ \quad \textbf{for } i \leftarrow 1 \textbf{ to } r \\ \quad \quad \textbf{do } \{ q_i \leftarrow ((j \bmod m^{r-i+1}) \bmod m^{r-i}) + 1 \\ \quad \quad \quad \textbf{if } i \notin \{q_1, \dots, q_r\} \\ \quad \quad \quad \quad \textbf{then } \mathcal{M}_{i,j}^r \leftarrow \text{sim}(\{R_{*,q_1}, \dots, R_{*,q_r}\}, R_{*,i}) \\ \quad \quad \quad \quad \textbf{else } \mathcal{M}_{i,j}^r \leftarrow 0 \\ \quad \quad \textbf{for } i \leftarrow 1 \textbf{ to } m \\ \quad \quad \textbf{do } \{ \textbf{if } \mathcal{M}_{i,j}^r \neq \text{among the } k \text{ largest values in } \mathcal{M}_{*,j}^r \\ \quad \quad \quad \textbf{then } \mathcal{M}_{i,j}^r \leftarrow 0 \end{array} \right\}$  (1)
    (2)
    (3)
  return  $\mathcal{M}^1, \mathcal{M}^2, \dots, \mathcal{M}^l$ 

```

**5.1.1 Itemset-Item Similarity.** As in the first-order model, the key step in the proposed higher-order item-based *top-N* recommendation algorithm is the method used to determine the similarity between an itemset and the various items of the dataset. In our scheme these similarities are computed by using relative straightforward extensions of the cosine- and conditional probability-based approaches described in Section 4.1.1.

Specifically, the similarity between an itemset  $\{q_1, q_2, \dots, q_r\}$  and an other item  $u$  is computed as follows. In the case of the cosine-based approach, we first

construct an  $n$ -element vector  $\vec{v}$  such that

$$\vec{v}(i) = \begin{cases} 0, & \text{if at least one of the } R_{i,j} = 0 \text{ for } j = 1, 2, \dots, r, \\ \sum_{j=1}^r \frac{R_{i,j}}{\|R_{*,q_j}\|_2}, & \text{otherwise.} \end{cases}$$

Essentially,  $\vec{v}$  is the sum of the individual unit-length normalized item-vectors of the items in the itemset with the added constraint that if a particular row of the matrix does not contain all  $r$  items it will be set to zero. Using this vector, the cosine similarity between the itemset represented by  $\vec{v}$  and the item  $u$  is computed using Eq. (1).

In the case of the conditional probability-based approach, the similarity is computed using an approach similar to Eq. (3) as follows:

$$\text{sim}(\{q_1, q_2, \dots, q_r\}, u) = \frac{\sum_{\forall i: R_{i,q_j} > 0, \text{for } j=1,2,\dots,r} R_{i,q_i}}{\text{Freq}(\{q_1, q_2, \dots, q_r\}) \times (\text{Freq}(u))^\alpha}. \quad (4)$$

Note that  $\text{Freq}(\{q_1, q_2, \dots, q_r\})$  is the number of rows in the matrix that contain all the items in the set. Also, since the rows of the user-item matrix  $R$  have been normalized to be of unit length  $R_{i,q_1} = R_{i,q_2} = \dots = R_{i,q_r}$ .

## 5.2 Applying the Model

The *top-N* recommendations for an active user are computed using the algorithm shown in Algorithm 5.2, where  $\mathcal{M}^1, \dots, \mathcal{M}^I$  are the different model matrices,  $U$  is the  $m \times 1$  vector storing the items that have already been purchased by the user, and  $N$  is the number of items to be recommended. The format of  $U$  and the format of the returned vector are identical to that used by the earlier item-to-item similarity algorithm (Algorithm 4.2).

**Algorithm 5.2:** `APPLYHIGHERORDERMODEL` ( $\mathcal{M}^1, \mathcal{M}^2, \dots, \mathcal{M}^l, U, N$ )

**for**  $r \leftarrow 1$  **to**  $l$  (1)

```

do {  

    for  $j \leftarrow 1$  to  $m^r$   

        do {  

            for  $i \leftarrow 1$  to  $r$   

                do {  

                    if  $U_{q_1} == 1$  and  $U_{q_2} == 1$  and ...  $U_{q_r} == 1$   

                        then  $U_j^r \leftarrow 1$   

                        else  $U_j^r \leftarrow 0$ 
                }
        }
}

```

$$X \leftarrow \sum_{r=1}^R \mathcal{M}^r U^r \quad (2)$$

**for**  $j \leftarrow 1$  **to**  $m$  (3)

**do** {  
**if**  $U_i \neq 0$   
**then**  $x_i \leftarrow 0$

**for**  $j \leftarrow 1$  **to**  $m$  (4)

**do** { if  $x_i \neq$  among the  $N$  largest values in  $x$   
then  $x_i \leftarrow 0$

**return** (x)

Algorithm 5.2 first generates  $I$  different vectors  $U^1, U^2, \dots, U^I$  of size  $m \times 1, m^2 \times 1, \dots, m^I \times 1$ , respectively (loop at line 1). For a particular value of  $r$ ,  $U^r$  is constructed by generating every possible combination of  $r$  items  $\{q_1, q_2, \dots, q_r\}$  and setting the corresponding entry of  $U^r$  to one if the active user has purchased all of these items and zero otherwise. Note that the row-index  $j$  of each itemset is constructed so that it is identical to the column-index of the same itemset used to populate the corresponding  $M^r$  matrix. The algorithm then computes the vector  $x$  by adding the various matrix-vector products of the corresponding  $M^r$  and  $U^r$  pairs (line 2). Finally, the algorithm proceeds to first filter out the items that the active user has already purchased (loop at line 3) and then retain only the  $N$  most similar items (loop at line 4).

### 5.3 Practical Considerations

Unfortunately, the higher-order item-based models described in the previous section are not computationally feasible because the model parameters that we need to compute and store (i.e., the  $k$  most similar items of the various itemsets) grows exponentially with the order of the model. Moreover, for most datasets, the occurrence frequency of many of these itemsets will be either zero or very small making it impossible to accurately estimate the  $k$  most similar items for each itemset. For this reason, our higher-order algorithms do not compute and store the itemset-to-item similarities for all itemsets but only for those itemsets that occur a sufficiently large number of times in the user-item matrix  $R$ . In particular, using the notion of *frequent itemsets* [Agrawal et al. 1993, 1996] developed by the data mining community, we use a computationally efficient algorithm [Seno and Karypis 2001] to find all frequent itemsets up to size  $I$  that occur in  $\sigma\%$  of the rows (i.e., transactions), and compute the  $k$  most similar other items only for these frequent itemsets. Note that the threshold  $\sigma$  is commonly referred to as the *minimum support constraint*.

The frequent-itemset based approach solves the issues associated with computational complexity but introduces two new problems. First, we need to develop a method that can be used to select the value of the minimum support constraint. A high value will result in a higher-order scheme that uses very few itemsets and as such it does not utilize its full potential, whereas a low value may lead to an exponentially large number of itemsets, making it computationally intractable. Unfortunately, there are no good ways to *a priori* select the value of support. This is because for a given value of  $\sigma$  the number of frequent itemsets that exist in a dataset depends on the dataset's density and the item co-occurrence patterns in the various rows. The same support value can lead to very few patterns in one dataset and a huge number of patterns in another. For this reason, selecting the right value of  $\sigma$  may require extensive experimentation to obtain a good balance between computational efficiency and *top-N* recommendation quality.

Second, since our higher-order models now only contain information about a small subset of the possible itemsets, there may be a number of itemsets that can be constructed from the items present in the active user's basket  $U$  that are not present in the model. One solution to this problem may be to just ignore

those itemsets while computing *top-N* recommendations. Such an approach is similar in spirit to some of the association rule-based *top-N* recommendation algorithms that are described in the related research section (Section 3) that have been shown to actually perform worse [Demiriz 2001] than the first-order item-based schemes described in Section 4. One of the reasons why such an approach may not be advisable is that if we consider the contributions that each item in  $U$  makes in determining the *top-N* recommended items, items that appear in frequent itemsets will tend to contribute more than items that do not. For example, an item that is present in a size-two and a size-three frequent itemset will have been used to determine the  $k$  most similar items of three different contributors to the final result (i.e., the  $k$ -most similar lists of the item itself and its size-two and size-three itemsets). However, an item that is not present in any frequent itemset will only contribute once to the final result. This creates an asymmetry on how the different items of a user's basket are used that can lead to relatively poor *top-N* recommendation performance.

For this reason, while computing the *top-N* recommendations for an active user we do not ignore any infrequent itemsets that it contains but use information from the first-order model to derive an approximation of its  $k$  most similar items. This is done as follows. For each infrequent itemset  $\{u, v, w\}$  that is derived from  $U$ , our algorithm treats it as a new basket and computes a *top-k* recommendation using the information from the first-order model (i.e., the  $k$  most similar items of each item). The weights associated with these *top-k* recommended items are scaled to be of unit length (for the same reasons discussed in Section 4.2) and are used as the  $k$ -most similar items for that itemset. Thus, by using such an approach our algorithm does not discriminate between items that are present in frequent itemsets and items that are not, while still maintaining the computational advantages of building higher-order models based only on frequent itemsets.

## 6. EXPERIMENTAL RESULTS

In this section, we experimentally evaluate the item-based *top-N* recommendation algorithms and compare their performance against the performance of the user-based *top-N* recommendation algorithm. All experiments were performed on a Intel Xeon based workstation running at 1.7GHz, 1GBytes of memory, and Linux-based operating system.

### 6.1 Experimental Design and Metrics

To evaluate the quality of the *top-N* recommendations, we split each of the datasets into a *training* and *test* set by randomly selecting one of the nonzero entries of each row to be part of the test set, and used the remaining entries for training.<sup>1</sup> For each user we obtained the *top-N* recommendations by using the items present in the training set as the *basket* for that user. In the case of the item-based algorithms, the *top-N* recommendations were computed using only the training set to build the item similarity models. Similarly, in the case of

---

<sup>1</sup>Our datasets were such that each row had at least two nonzero entries.

the user-based algorithms, the nearest neighbors and *top-N* recommendations were computed only using the training set.

The quality was measured by looking at the number of *hits* and their position within the *top-N* items that were recommended by a particular scheme. The number of hits is the number of items in the test set that were also present in the *top-N* recommended items returned for each user. We computed two quality measures that we will refer to them as the *hit-rate* (HR) and the *average reciprocal hit-rank* (ARHR) that are defined as follows. If  $n$  is the total number of customers/users, the hit-rate of the recommendation algorithm was computed as:

$$\text{hit-rate (HR)} = \frac{\text{Number of hits}}{n}. \quad (5)$$

An HR value of 1.0 indicates that the algorithm was able to always recommend the hidden item, whereas an HR value of 0.0 indicates that the algorithm was not able to recommend any of the hidden items. One limitation of the hit-rate measure is that it treats all hits equally regardless of where they appear in the list of the *top-N* recommended items. That is, a hit that occurs in the first position is treated equally with a hit that occurs in the  $N$ th position. This limitation is addressed by the average reciprocal hit-rank measure that rewards each hit based on where it occurred in the *top-N* list. If  $h$  is the number of hits that occurred at positions  $p_1, p_2, \dots, p_h$  within the *top-N* lists (i.e.,  $1 \leq p_i \leq N$ ), then the average reciprocal hit-rank is equal to

$$\text{average reciprocal hit-rank (ARHR)} = \frac{1}{n} \sum_{i=1}^h \frac{1}{p_i}. \quad (6)$$

That is, hits that occur earlier in the *top-N* lists are weighted higher than hits that occur later in the list. The highest value of ARHR is equal to the hit-rate and occurs when all the hits occur in the first position, whereas the lowest value of the ARHR is equal to  $\text{hit-rate}/N$  when all the hits occur in the last position in the list of the *top-N* recommendations.

In order to ensure that our results are not sensitive to the particular training-test partitioning of each dataset, for each of the experiments we performed ten different runs, each time using a different random partitioning into training and test sets. The results reported in the rest of this section are the averages over these ten trials. Furthermore, to better compare the various results we used two different statistical tests to compare the averages obtained from the ten different random partitionings that are based on the paired *t*-test for pairwise comparisons and on the Bonferroni test for multiple comparisons. Both tests were performed at a 95% confidence interval.

Finally, in all of experiments we used  $N = 10$  as the number of items top be recommended by the *top-N* recommendation algorithms.

## 6.2 Evaluation on Real Datasets

We evaluated the performance of the different *top-N* recommendation algorithms using eight different datasets whose characteristics are shown in Table I. For each dataset, this table shows the number of users, the number

Table I. The Characteristics of the Various Datasets used in Evaluating the *top-N* Recommendation Algorithms

| Name  | Number of Users | Number of Items | Number of Transactions | Density | Average Basket Size |
|-------|-----------------|-----------------|------------------------|---------|---------------------|
| ctlg1 | 58565           | 502             | 209715                 | 0.71%   | 3.58                |
| ctlg2 | 23480           | 55879           | 1924122                | 0.15%   | 81.95               |
| ctlg3 | 58565           | 39080           | 453219                 | 0.02%   | 7.74                |
| ccard | 42629           | 68793           | 398619                 | 0.01%   | 9.35                |
| ecmrc | 6667            | 17491           | 91222                  | 0.08%   | 13.68               |
| em    | 8002            | 1648            | 769311                 | 5.83%   | 96.14               |
| ml    | 943             | 1682            | 100000                 | 6.31%   | 106.04              |
| skill | 4374            | 2125            | 82612                  | 0.89%   | 18.89               |

of items, and the total number of transactions (i.e., nonzeros in the resulting user-item matrix). In addition, the column labeled “Density” shows the percentage of nonzero entries in the user-item matrix and the column labeled “Avg. Basket Size” shows that average number of items in each transaction.

These datasets can be broadly classified into two categories. The first category (containing the first five datasets) was derived from customer purchasing transactions and is typical of datasets arising in e-commerce and traditional marketing applications of *top-N* recommender systems. Specifically, the *ctlg1*, *ctlg2*, and *ctlg3* datasets correspond to the catalog purchasing transactions of two major mail-order catalog retailers. Note that *ctlg1* and *ctlg3* correspond to the same set of transactions but they differ on what constitutes an item. The items of *ctlg3* correspond to individual products, whereas the items of *ctlg1* correspond to the top-level product categories, that is, a particular nonzero entry in the user-item matrix is a transaction indicating that a particular user has purchased an item from a particular product category. The *ecmrc* dataset corresponds to web-based purchasing transactions of an e-commerce site. The *ccard* dataset corresponds to credit card purchasing transactions of a major department store’s credit card.

The second category (containing the remaining datasets) was obtained from two different application areas and corresponds to non-traditional uses of *top-N* recommender systems. In particular, the *em* and *ml* datasets correspond to movie ratings and were obtained from the *EachMovie* [McJones and DeTreville 1997] and the *MovieLens* [MovieLens 2003] research projects, respectively. Note that these two datasets contain multi-value ratings that indicate how much each user liked a particular movie or not. For the purpose of our experiments we ignored the values of these ratings and treated them as an indication that the user has seen that particular movie. By performing this conversion we focus on the problem of predicting whether or not a user will see a particular movie and not whether or not he or she will like it. Finally, the *skill* dataset corresponds to the information technology related skills that are present in the resumes of various individuals and were obtained from a major online job portal. The *top-N* recommendation problem in this dataset is that of predicting a set of other related skills that can potentially act as a suggestion to the user on how to improve his or her career.

Table II. The Effect of the Similarity Normalization on the Recommendation Quality Achieved by the First-Order Cosine- and Conditional Probability-Based Recommendation Algorithms

|       | Top-10 Hit-Rate |        |                   |        | Top-10 Average Reciprocal Hit-Rank |        |                   |        |
|-------|-----------------|--------|-------------------|--------|------------------------------------|--------|-------------------|--------|
|       | Cosine          |        | Cond. Probability |        | Cosine                             |        | Cond. Probability |        |
|       | SNorm+          | SNorm- | SNorm+            | SNorm- | SNorm+                             | SNorm- | SNorm+            | SNorm- |
| ctlg1 | <b>0.406</b>    | 0.396  | <b>0.415</b>      | 0.404  | <b>0.208</b>                       | 0.203  | <b>0.213</b>      | 0.206  |
| ctlg2 | <b>0.147</b>    | 0.143  | <b>0.154</b>      | 0.127  | <b>0.070</b>                       | 0.069  | <b>0.074</b>      | 0.064  |
| ctlg3 | <b>0.534</b>    | 0.529  | <b>0.540</b>      | 0.515  | <b>0.315</b>                       | 0.310  | <b>0.320</b>      | 0.303  |
| ccard | <b>0.162</b>    | 0.160  | <b>0.176</b>      | 0.167  | <b>0.119</b>                       | 0.118  | <b>0.130</b>      | 0.126  |
| ecmrc | <b>0.170</b>    | 0.166  | 0.174             | 0.174  | <b>0.096</b>                       | 0.093  | 0.098             | 0.097  |
| em    | <b>0.407</b>    | 0.400  | <b>0.405</b>      | 0.395  | <b>0.189</b>                       | 0.186  | <b>0.189</b>      | 0.183  |
| ml    | <b>0.271</b>    | 0.264  | <b>0.272</b>      | 0.249  | <b>0.119</b>                       | 0.115  | <b>0.119</b>      | 0.110  |
| skill | <b>0.370</b>    | 0.358  | <b>0.373</b>      | 0.313  | <b>0.178</b>                       | 0.172  | <b>0.178</b>      | 0.151  |

Bold-faced entries correspond to schemes that perform statistically better at 95% confidence interval using the paired  $t$ -test.

**6.2.1 Parameter Evaluation.** Since there are a number of alternative options that control the various aspects of the proposed item-based *top-N* recommendation algorithm, it is not possible to provide an exhaustive comparison of all possible combinations without making this article unduly large. Instead, we provide comparisons of different alternatives for each option after making a reasonable choice for the other options.

**6.2.1.1 Effect of Similarity Normalization.** Our first experiment was designed to evaluate the effect of the similarity normalization that is discussed in Section 4.2. Table II shows the HR and ARHR results achieved by four different item-based recommendation algorithms. Two of them use the cosine as the similarity function whereas the other two use the conditional probability. The difference between each pair of algorithms is that one does not normalize the similarities (those labeled “SNorm-”) whereas the other normalizes them (those labeled “SNorm+”). For all four algorithms, the rows of the matrix were normalized so that they are of unit length,  $k$  (the number of nearest items to use in the model) was set to 20, and a value of  $\alpha = 0.5$  was used for the schemes that are based on the conditional probability-based approach. In addition, all of these schemes correspond to first-order item-based models.

Looking at the results in Table II, we can see that the algorithms that use similarity normalization achieve better results (both in terms of HR and ARHR) compared to their counterparts that do not use such normalization. As we can see from these results, in all cases the scheme that normalizes the similarity values performs better than the scheme that does not. The actual improvement is dataset and algorithm dependent. In general, the relative improvements tend to be higher for the conditional probability based scheme than the cosine-based scheme. On average, the HR of the cosine-based scheme improves by 2.15%, whereas the HR of the conditional probability-based scheme improves by 7.85%. Similar trends are observed when comparing the performance of the various algorithms using the ARHR measure. To ensure that these differences

Table III. The Effect of Row Normalization on the Recommendation Quality Achieved by the Cosine- and Conditional Probability-Based Recommendation Algorithms

|       | Top-10 Hit-Rate |              |                   |              | Top-10 Average Reciprocal Hit-Rank |              |                   |              |
|-------|-----------------|--------------|-------------------|--------------|------------------------------------|--------------|-------------------|--------------|
|       | Cosine          |              | Cond. Probability |              | Cosine                             |              | Cond. Probability |              |
|       | RNorm+          | RNorm-       | RNorm+            | RNorm-       | RNorm+                             | RNorm-       | RNorm+            | RNorm-       |
| ctlg1 | 0.406           | 0.406        | <b>0.415</b>      | 0.406        | 0.208                              | 0.208        | <b>0.213</b>      | 0.208        |
| ctlg2 | <b>0.147</b>    | 0.143        | <b>0.154</b>      | 0.143        | 0.070                              | 0.069        | <b>0.074</b>      | 0.069        |
| ctlg3 | 0.534           | <b>0.536</b> | <b>0.540</b>      | 0.536        | 0.315                              | <b>0.317</b> | <b>0.320</b>      | 0.317        |
| ccard | 0.162           | <b>0.179</b> | 0.176             | <b>0.179</b> | 0.119                              | <b>0.132</b> | 0.130             | <b>0.132</b> |
| ecmrc | 0.170           | <b>0.173</b> | <b>0.174</b>      | 0.173        | 0.096                              | <b>0.097</b> | 0.098             | 0.097        |
| em    | <b>0.407</b>    | 0.395        | <b>0.405</b>      | 0.395        | <b>0.189</b>                       | 0.186        | <b>0.189</b>      | 0.186        |
| ml    | <b>0.271</b>    | 0.261        | <b>0.272</b>      | 0.260        | <b>0.119</b>                       | 0.112        | <b>0.119</b>      | 0.112        |
| skill | <b>0.370</b>    | 0.344        | <b>0.373</b>      | 0.344        | <b>0.178</b>                       | 0.165        | <b>0.178</b>      | 0.165        |

For each experiment, bold-faced entries correspond to schemes that perform statistically better using the paired *t*-test.

are statistically significant we tested them using the paired *t*-test. Table II highlights with a bold-faced font the HR and ARHR entries of the scheme that is significantly better than the others. As we can see from these results, for all datasets, the differences are indeed statistically significant. Due to this performance advantage in the rest of our experiments, we will always use similarity normalization.

**6.2.1.2 Effect of Row Normalization.** The second experiment was designed to evaluate the effect of row-normalization so that customers that purchase many items will weigh less during the item similarity calculations. Table III shows the HR and ARHR achieved by four different item-based recommendation algorithms. Two of them use the cosine as the similarity function whereas the other two use the conditional probability. The difference between each pair of algorithms is that one does not normalize the rows (those labeled “RNorm–”) whereas the other normalizes them (those labeled “RNorm+”). Also, the entries in Table III that correspond to the schemes that perform statistically better based on the paired *t*-test are highlighted using a bold-faced font. For all experiments *k* was set to 20, and for the two conditional probability-based algorithms, a value of  $\alpha = 0.5$  was used. In addition, all of these schemes correspond to first-order item-based models.

From the results in Table III we can see that on average, the row-normalized version performs somewhat better for both the cosine- and the conditional probability-based schemes. Specifically, the average improvement in terms of HR for all eight datasets is 0.69% for the cosine- and 3.05% for conditional probability-based scheme. Similar observations can be made by looking at the ARHR results as well. Comparing the statistical significance of these results, we can see that for the conditional probability-based scheme the scheme that normalizes the rows performs statistically better on seven out of the eight datasets. However, in the case of the cosine-based scheme there is a certain amount of variation in which scheme performs statistically better, and some of these improvements are not statistically significant. Because of these improvements in the rest of our experiments we will always use row normalization.

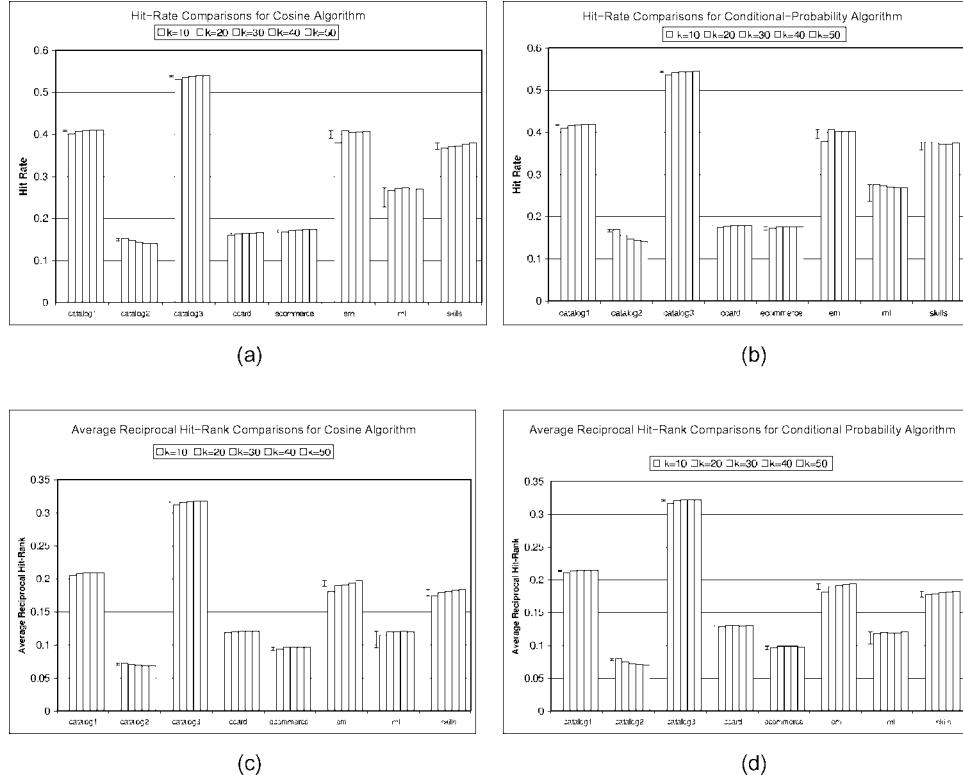


Fig. 1. The HR and ARHR as a function of the number of most similar items ( $k$ ) used in computing the  $top-N$  recommendations for the cosine- and conditional probability-based recommendation algorithms. The error bars associated with each dataset correspond to the minimum required difference in either HR or ARHR in order for two schemes to be statistically different.

**6.2.1.3 Model Size Sensitivity.** Recall from Section 4.1 that the item-based recommendations are computed using a model that utilizes the  $k$  most similar items for each one of the different items. To evaluate the sensitivity of the different algorithms on the value of  $k$  we performed an experiment in which we let  $k$  take the values of 10, 20, 30, 40, and 50. The recommendation performance in terms of HR and ARHR for these experiments is shown in Figure 1 for the cosine- and conditional probability-based algorithms. For each dataset Figure 1 also shows the minimum required difference in the respective performance measure in order for a particular value of  $k$  to perform significantly better (or worse) than the remaining  $k$  values. These differences were computed using the Bonferroni test and are shown using the error-bars. For both classes of algorithms we used the first-order models and in the case of the conditional probability-based schemes the experiments were performed using a value of  $\alpha = 0.5$ .

As we can see from these experiments, the overall recommendation accuracy of the item-based algorithms does tend to improve as we increase the value of  $k$ . The only exception is the *ctlg2* dataset for which both the HR and the

ARHR tend to consistently decrease as we increase  $k$ . Overall, the average HR for the cosine-based algorithm improves by 1.8% as we vary  $k$  from 10 to 50 items; whereas in the case of the conditional probability-based algorithm the average improvement in HR is 0.65%. Similar minor improvements are achieved in terms of ARHR as well. However, as the figure illustrates, most of these improvements are not statistically significant and no particular value of  $k$  dominates the rest. These results indicate that (i) even for small values of  $k$  the item-based recommendation algorithms provide reasonably accurate recommendations; and (ii) increasing the value of  $k$  does not lead to significant improvements. This is particularly important since small values of  $k$  lead to fast recommendation rates (i.e., low computational requirements) without materially affecting the overall quality of the recommendations. Note that the diminishing incremental improvements achieved by increasing the value of  $k$  are a direct consequence of the fact that we are only looking for 10 recommended items (i.e.,  $N = 10$ ). As a result, once  $k$  is sufficiently large, to ensure that the various item-to-item lists have sufficient common items, any further increases in  $k$  will not change the order of the  $top-N$  items.

**6.2.1.4 Item Frequency Scaling Sensitivity.** One of the parameters of the conditional probability-based  $top-N$  recommendation algorithm is the value of  $\alpha$  used to control the extent to which the similarity to frequently purchased items will be de-emphasized. To study the sensitivity of the recommendation algorithm on this parameter we performed a sequence of experiments in which we varied  $\alpha$  from 0.0 to 1.0 in increments of 0.1. Figure 2 shows the HR and ARHR achieved on the various datasets for the different values of  $\alpha$ . As with the results of the previous study, the minimum required differences computed using the Bonferroni test are shown using error bars. Note that these results were obtained using the first-order item-based model and  $k = 20$ .

From these results we can see that for all datasets the value of  $\alpha$  has a significant impact on the recommendation quality, as different values of  $\alpha$  lead to substantially different values of HR and ARHR. Despite this variability, for almost all datasets, if  $0.3 \leq \alpha \leq 0.6$ , then the conditional probability-based scheme achieves consistently good performance. Also note that as we increase the value of  $\alpha$ , the changes in the HR and ARHR are fairly smooth, and follow a  $\cap$ -shaped curve. This suggests that the optimal value of  $\alpha$  can be easily estimated for each particular dataset by hiding a portion of the training set and using it to find the value of  $\alpha$  that leads to the highest HR or ARHR. Moreover, since the values of  $\alpha$  that lead to both the highest HR or ARHR values are consistent for most of the datasets, we can learn the value of  $\alpha$  that optimizes one of the two measures as it will also lead to optimal or near-optimal performance with respect to the other measure.

A further study of the values of  $\alpha$  that lead to the highest HR and ARHR values and the properties of the various datasets used in our experiment reveal another interesting trend. If we compare the highest HR value to the HR value achieved for  $\alpha = 0.0$  we see that for *ctlg1*, *ccard*, *ecmrc*, and *ctlg3*, the highest value is usually less than 3.2% better than that for  $\alpha = 0.0$ . On the other hand, the improvement for *skill*, *em*, *ctlg2*, and *ml* ranges from 16% to 91%.

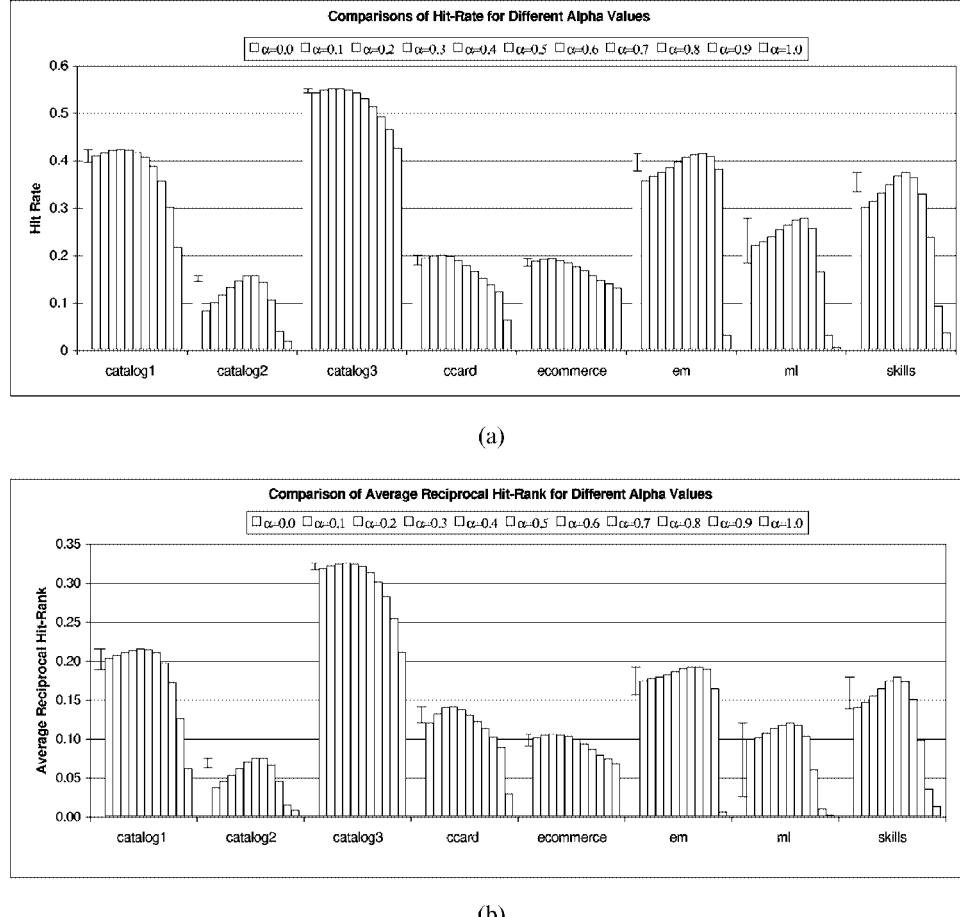


Fig. 2. The HR and ARHR as a function of the item-frequency-based scaling achieved by the  $\alpha$  parameter for conditional probability-based recommendation algorithms. The error bars associated with each dataset correspond to the minimum required difference in either HR or ARHR in order for two schemes to be statistically different.

Similar trends can be observed by focusing on the ARHR measure. Thus, there is a group of datasets for which there is a clear benefit in trying to optimize the value of  $\alpha$ . Moreover, the datasets for which we achieve significant HR (or ARHR) improvements are those datasets that according to the statistics shown in Table I have some of the highest densities and the largest number of items per user.

**6.2.1.5 Model Order Sensitivity.** Our experiments so far focused on first-order item-based *top-N* recommendation algorithms. However, as discussed in Section 5, both the cosine- and the conditional probability-based schemes can be extended to higher order-models by using frequent itemsets of different length and using an interpolating approach to combine the recommendations performed by the different models. Table IV shows the HR and the ARHR results

Table IV. The Recommendation Quality as a Function of the Order of the Model that is Used

| Top-10 Hit-Rate |              |       |       |       |              |       |       |              |              |        |
|-----------------|--------------|-------|-------|-------|--------------|-------|-------|--------------|--------------|--------|
| Name            | $\sigma$ (%) | $F_2$ | $F_3$ | Cos1  | Cos2         | Cos3  | CPrb1 | CPrb2        | CPrb3        | RqDiff |
| ctlg1           | 0.1          | 868   | 486   | 0.406 | 0.405        | 0.406 | 0.415 | 0.414        | 0.416        | 0.0039 |
| ctlg2           | 5.0          | 764   | 835   | 0.147 | 0.147        | 0.147 | 0.154 | 0.154        | 0.154        | 0.0048 |
| ctlg3           | 0.05         | 2150  | 1437  | 0.534 | 0.535        | 0.535 | 0.540 | 0.540        | 0.540        | 0.0022 |
| ccard           | 0.01         | 3326  | 2056  | 0.162 | 0.162        | 0.162 | 0.176 | 0.175        | 0.175        | 0.0022 |
| ecmrc           | 0.01         | 255   | 112   | 0.170 | 0.170        | 0.170 | 0.174 | 0.174        | 0.174        | 0.0094 |
| em              | 20.0         | 4077  | 52434 | 0.407 | <u>0.419</u> | 0.416 | 0.405 | <u>0.418</u> | <u>0.415</u> | 0.0108 |
| ml              | 10.0         | 9921  | 87090 | 0.271 | 0.267        | 0.270 | 0.272 | 0.279        | 0.275        | 0.0284 |
| skill           | 1.0          | 4485  | 16820 | 0.370 | 0.361        | 0.367 | 0.373 | 0.380        | 0.379        | 0.0191 |

| Top-10 Average Reciprocal Hit-Rank |              |       |       |       |              |              |       |              |              |        |
|------------------------------------|--------------|-------|-------|-------|--------------|--------------|-------|--------------|--------------|--------|
| Name                               | $\sigma$ (%) | $F_2$ | $F_3$ | Cos1  | Cos2         | Cos3         | CPrb1 | CPrb2        | CPrb3        | RqDiff |
| ctlg1                              | 0.1          | 868   | 486   | 0.208 | 0.208        | 0.208        | 0.213 | 0.213        | 0.214        | 0.0037 |
| ctlg2                              | 5.0          | 764   | 835   | 0.070 | 0.070        | 0.070        | 0.074 | 0.074        | 0.074        | 0.0035 |
| ctlg3                              | 0.05         | 2150  | 1437  | 0.315 | 0.316        | 0.315        | 0.320 | 0.320        | 0.321        | 0.0027 |
| ccard                              | 0.01         | 3326  | 2056  | 0.119 | 0.118        | 0.119        | 0.130 | 0.128        | 0.129        | 0.0013 |
| ecmrc                              | 0.01         | 255   | 112   | 0.096 | 0.095        | 0.096        | 0.098 | 0.098        | 0.098        | 0.0031 |
| em                                 | 20.0         | 4077  | 52434 | 0.189 | <u>0.201</u> | <u>0.200</u> | 0.189 | <u>0.199</u> | <u>0.197</u> | 0.0052 |
| ml                                 | 10.0         | 9921  | 87090 | 0.119 | 0.114        | 0.118        | 0.119 | 0.120        | 0.119        | 0.0143 |
| skill                              | 1.0          | 4485  | 16820 | 0.178 | 0.172        | 0.176        | 0.178 | 0.184        | 0.184        | 0.0130 |

Underlined entries correspond to the higher-order schemes that perform statistically better than the corresponding first-order scheme.

obtained by using such higher-order interpolated models for both the cosine- and the conditional probability-based approaches. In particular, Table IV shows the results obtained by a first-, second-, and third-order interpolated models. Note that the first-order model results are identical to those presented in the previous sections.

One of the key parameters of higher-order models is the support threshold ( $\sigma$ ) used by the frequent pattern discovery algorithm to identify the frequent itemsets to be used in the models. We used different values of the support threshold for each dataset depending on the density and the degree to which different items co-occur in the different datasets. These values are shown in the second column of Table IV. They were selected so that (i) they lead to a reasonable number of frequent itemsets and (ii) each frequent itemset has a sufficiently large support to ensure the statistical significance of the similarities that are computed between an itemset and the remaining items. The actual number of frequent size-two and size-three frequent itemsets that were discovered and used to build the interpolated second- and third-order models are shown in the columns labeled " $F_2$ " and " $F_3$ ", respectively. The last column in these tables (labeled "Reqd. Diff") shows the minimum difference of the respective performance measure that is required in order for two schemes to be statistically different from each other using the Bonferroni test. For all experiments  $k$  was set to 20, and for the conditional probability-based algorithms we used a value of  $\alpha = 0.5$ .

As we can see from these results, higher-order item-based models do not lead to any significant improvements in either HR or ARHR. For most datasets, the results obtained across the different schemes (i.e., 1st-, 2nd-, and 3rd-order

Table V. The Quality of the Recommendations Obtained by the Naive, the Item-Based, and the User-Based Recommendation Algorithm

| Top-10 Hit-Rate |       |          |        |                       |                              |
|-----------------|-------|----------|--------|-----------------------|------------------------------|
|                 | User  | Frequent | Cosine | CProb- $\alpha = 0.5$ | CProb- $\alpha = \text{Opt}$ |
| ctlg1           | 0.398 | 0.215    | 0.406  | 0.415                 | 0.421                        |
| ctlg2           | 0.150 | 0.025    | 0.147  | 0.154                 | 0.155                        |
| ctlg3           | 0.494 | 0.030    | 0.534  | 0.540                 | 0.549                        |
| ccard           | 0.158 | 0.079    | 0.162  | 0.176                 | 0.198                        |
| ecmrc           | 0.178 | 0.029    | 0.170  | 0.174                 | 0.191                        |
| em              | 0.453 | 0.367    | 0.407  | 0.405                 | 0.412                        |
| ml              | 0.281 | 0.131    | 0.271  | 0.272                 | 0.276                        |
| skill           | 0.384 | 0.238    | 0.370  | 0.373                 | 0.373                        |

| Top-10 Average Reciprocal Hit-Rank |       |          |        |                       |                              |
|------------------------------------|-------|----------|--------|-----------------------|------------------------------|
|                                    | User  | Frequent | Cosine | CProb- $\alpha = 0.5$ | CProb- $\alpha = \text{Opt}$ |
| ctlg1                              | 0.206 | 0.080    | 0.208  | 0.213                 | 0.214                        |
| ctlg2                              | 0.076 | 0.009    | 0.070  | 0.074                 | 0.074                        |
| ctlg3                              | 0.298 | 0.010    | 0.315  | 0.320                 | 0.324                        |
| ccard                              | 0.119 | 0.066    | 0.119  | 0.130                 | 0.140                        |
| ecmrc                              | 0.095 | 0.012    | 0.096  | 0.098                 | 0.105                        |
| em                                 | 0.221 | 0.169    | 0.189  | 0.189                 | 0.191                        |
| ml                                 | 0.128 | 0.046    | 0.119  | 0.119                 | 0.119                        |
| skill                              | 0.189 | 0.091    | 0.178  | 0.178                 | 0.178                        |

models) are very similar or within less than 1% of each other, which according to the Bonferroni test is not statistically significant. The only datasets for which higher-order models, and the second-order model in particular, did somewhat better than the first-order model are the *em*, *ml*, and *skill*/datasets. In particular, the second-order model improved the HR in the above datasets by 1.8% to 3.2%, and the ARHR by 3.3% to 6.3%. Also note that these three datasets are the ones that contain the most size-two and size-three frequent itemsets, suggesting that when a particular dataset contains a sufficient number of frequent itemsets, the higher-order models can improve the quality of the *top-N* recommendations. However, among these datasets, only the improvements achieved for the *em* dataset (corresponding to the underlined entries) are statistical significant.

**6.2.2 Overall Comparisons.** To compare the performance of the various item-based recommendation algorithms against each other and with that achieved by user-based algorithms we performed an experiment in which we computed the *top-N* recommendations using both the item-based and the user-based recommendation algorithms. The results from these experiments are shown in Table V. The user-based recommendations were obtained using the algorithm described in Herlocker et al. [1999] and Sarwar et al. [2000] with user-neighborhoods of size 50 and unit-length normalized rows. We used a similarity-weighted approach to determine the frequency of each item, and we did not include neighbors that had an identical set of items as the active item (as these neighbors do not contribute at all in the recommendation).

Table V includes three different sets of item-based results obtained with  $k = 20$ . The results labeled “Cosine” correspond to the cosine-based results.

The results labeled “CProb- $\alpha = 0.5$ ” correspond to the conditional probability-based algorithm in which  $\alpha$  was set to 0.5. The results labeled “CProb- $\alpha = \text{Opt}$ ” correspond to the conditional probability-based algorithm that uses the value of  $\alpha$  that achieved the highest performance in the experiments discussed in Section 6.2.1 for each dataset. All the item-based results were obtained using the first-order models. Table V also includes the *top-N* recommendation quality achieved by the naive algorithm, labeled “Frequent”, which recommends the  $N$  most frequent items not already present in the active user’s set of items.

Comparing the performance achieved by the item-based schemes with that achieved by the user-based scheme we can see that the cosine-based scheme performs better than the user-based scheme in three out of the eight datasets, whereas the conditional probability-based schemes that use  $\alpha = 0.5$  and  $\alpha = \text{Opt}$  outperform the user-based scheme in four out of eight and five out of eight datasets, respectively. On average, the cosine-based scheme does 1.15% and 4.04% worse than the user-based scheme in terms of HR and ARHR, respectively; the conditional probability-based scheme with  $\alpha = 0.5$  does 1.10% better and 1.30% worse than the user-based scheme in terms of HR and ARHR, respectively; whereas the conditional probability-based scheme with the best choice for  $\alpha$  does 4.65% and 1.29% better than the user-based scheme in terms of HR and ARHR, respectively. In general, all three item-based schemes seem to do worse than the user-based scheme for the denser datasets (e.g., *skill*, *em*, and *ml*), and do better for the sparser datasets (e.g., *ccard*, *ecmrc*, and *ctlg3*). Also the performance of the item-based schemes relative to the user-based scheme is somewhat worse when measured in terms of ARHR instead of HR. This suggests that in the case of user-based schemes the hidden items (i.e., hits) occur earlier in the list of *top-N* recommended items, even if in some cases the aggregate number hidden items that were able to recommend is smaller than the total number recommended by the item-based schemes.

Comparing the results achieved by the various item-based schemes we can see that the schemes based on conditional probability perform better than those based on cosine similarity. On average, in terms of HR, the conditional probability-based scheme with  $\alpha = 0.5$  does 2.5% better than the cosine-based scheme, whereas the scheme using the optimal value of  $\alpha$  performs 5.9% better. Finally, both the user- and item-based algorithms produce recommendations whose quality is substantially better than the recommendations produced by the naive “Frequent” algorithm.

To ensure that the above comparisons are significant we used the paired *t*-test to determine the number of datasets in which one scheme outperforms the other at a confidence interval of 95%. The results of this analysis are shown in Table VI for both the HR and the ARHR measures. Each entry in these two  $5 \times 5$  tables contains three numbers that correspond to the number of datasets in which the scheme corresponding to the row performed statistically better, the same, or worse than the scheme corresponding to the column, respectively. As we can see from these results, in almost all cases, the performance differences between each pair of schemes are statistically significant.

Table VI. Statistical Significance Comparisons of the Various *top-N* Recommendation Algorithms Using the Paired *t*-Test

| Top-10 Hit-Rate                    |         |          |         |                       |                              |
|------------------------------------|---------|----------|---------|-----------------------|------------------------------|
|                                    | User    | Frequent | Cosine  | CProb- $\alpha = 0.5$ | CProb- $\alpha = \text{Opt}$ |
| User                               | —       | 8, 0, 0  | 5, 0, 3 | 4, 0, 4               | 3, 0, 5                      |
| Frequent                           | 0, 0, 8 | —        | 0, 0, 8 | 0, 0, 8               | 0, 0, 8                      |
| Cosine                             | 3, 0, 5 | 8, 0, 0  | —       | 1, 1, 6               | 0, 0, 8                      |
| CProb- $\alpha=0.5$                | 4, 0, 4 | 8, 0, 0  | 6, 1, 1 | —                     | 0, 1, 7                      |
| CProb- $\alpha=\text{Opt}$         | 5, 0, 3 | 8, 0, 0  | 8, 0, 0 | 7, 1, 0               | —                            |
| Top-10 Average Reciprocal Hit-Rank |         |          |         |                       |                              |
|                                    | User    | Frequent | Cosine  | CProb- $\alpha = 0.5$ | CProb- $\alpha = \text{Opt}$ |
| User                               | —       | 8, 0, 0  | 4, 1, 3 | 4, 0, 4               | 4, 0, 4                      |
| Frequent                           | 0, 0, 8 | —        | 0, 0, 8 | 0, 0, 8               | 0, 0, 8                      |
| Cosine                             | 3, 1, 4 | 8, 0, 0  | —       | 1, 2, 5               | 0, 2, 6                      |
| CProb- $\alpha=0.5$                | 4, 0, 4 | 8, 0, 0  | 5, 2, 1 | —                     | 0, 3, 5                      |
| CProb- $\alpha=\text{Opt}$         | 4, 0, 4 | 8, 0, 0  | 6, 2, 0 | 5, 3, 0               | —                            |

The three numbers in each cell show the number of datasets in which the scheme corresponding to the row performed statistically better, the same, or worse than the scheme corresponding to the column.

Table VII. The Computational Requirements for Computing the *top-N* Recommendations for Both the User- and Item-Based Algorithms

| Name  | User-based |          | Item-based |          |          |
|-------|------------|----------|------------|----------|----------|
|       | RcmdTime   | RcmdRate | ModelTime  | RcmdTime | RcmdRate |
| ctlg1 | 62.68      | 934      | 0.10       | 0.16     | 366031   |
| ctlg2 | 83.53      | 281      | 19.35      | 1.82     | 12901    |
| ctlg3 | 13.57      | 4315     | 0.69       | 0.78     | 75083    |
| ccard | 17.59      | 2427     | 0.98       | 0.79     | 53960    |
| ecmrc | 0.48       | 13889    | 0.10       | 0.08     | 83337    |
| em    | 49.25      | 162      | 1.74       | 0.33     | 24248    |
| ml    | 0.46       | 2049     | 0.24       | 0.05     | 18859    |
| skill | 1.64       | 2667     | 0.13       | 0.07     | 62485    |

**6.2.2.1 Computational Requirements.** One of the advantages of the item-based algorithm is that it has much smaller computational requirements than the user-based *top-N* recommendation algorithm. Table VII shows the amount of time required by the two algorithms to compute the *top-N* recommendations for each one of the eight datasets. The column labeled “ModelTime” shows the amount of time required to build the item-based recommendation model (i.e., compute the  $k$  most similar items), the columns labeled “RcmdTime” show the amount of time required to compute the  $n$  recommendations for each one of the datasets, and the columns labeled “RcmdRate” show the rate at which the *top-N* recommendations were computed in terms of *recommendations/second*. Note that our implementation of the user-based *top-N* recommendation algorithm takes advantage of the sparse user-item matrix, and uses inverted indices in order to identify the nearest users as quickly as possible. All the times in Table VII are in seconds.

Looking at the results of Table VII we can see that the recommendation rates achieved by the item-based algorithm are 6 to 391 times higher than those achieved by the user-based algorithm. If we add the various “RcmdTime” for all eight data sets we can see that the overall recommendation rate for the item-based algorithm is 56715 recommendations/second compared to only 930 recommendations/second achieved by the user-based algorithm. This translates to one recommendation every 17  $\mu$ s for the item-based algorithm, versus 1075  $\mu$ s for the user-based algorithm. Also, as discussed in Section 4.3, the amount of time required to build the models for the item-based algorithm is quite small. In particular, even accounting for the model building time, the item-based algorithm is still 2 to 240 times faster than the user-based algorithm. Note that the reason that the user-based scheme is still slower even when we take into account the time required to build the models is the fact that the resulting user-user similarity matrix that needs to be computed is much denser than the corresponding item-item similarity matrix. This is because the density of the user-user similarity matrix depends on the existence of some frequently purchased items (i.e., dense columns in the matrix) which happens quite often, whereas in the case of the item-item similarity matrix, it is rare to have any dense rows (i.e., users that have purchased most of the items).

### 6.3 Evaluation on Synthetic Datasets

The performance of recommender systems is highly dependent on various characteristics of the dataset such as the number of items, the number of users, its sparsity, and the behavioral variability of the various users in terms of the items they buy/see. Furthermore, as the results in Section 6.2.2 have shown, the relative performance of various *top-N* recommendation algorithms do not vary uniformly across different datasets and it is quite likely that a particular scheme will outperform the rest for a particular dataset, whereas the same scheme might underperform when the dataset characteristics are changed. This dataset-specific behavior of recommendation schemes makes it hard to decide the best scheme for a particular application. The goal of this section is to study the influence of two key dataset characteristics, *sparsity* and *user's behavioral variability*, on the performance of the recommendation system and gain some insights as to which *top-N* recommendation algorithm is better-suited to which characteristics of a dataset. We conduct this study on synthetically generated datasets as they provide us the flexibility to individually isolate a dataset characteristic and vary its value while keeping the other characteristics constant.

We make use of the IBM synthetic dataset generator [Agrawal and Srikant 1994], which is widely used to mimic the transactions in the retail environment. The dataset generator is based on the observation that people tend to make purchases in sets of items. For example, if a user's basket contains {pillow covers, sheets, comforter, milk, bread, eggs}, then it can be thought of as made of two sets of items, the first set consists of items {pillow covers, sheets, comforter} and the second set is made of {milk, bread, eggs}. This set of items is referred as *itemset*. It is observed that the size of such *itemsets* is clustered around a

Table VIII. Parameters Taken by Synthetic Dataset Generator

| Description                   | Symbol   | IBM Symbol      | Value                   |
|-------------------------------|----------|-----------------|-------------------------|
| Number of users               | $n$      | $ \mathcal{D} $ | 5000                    |
| Number of items               | $m$      | $N$             | 1000                    |
| Average size of user's basket | $S_u$    | $ \mathcal{T} $ | 15, 30, & 45            |
| Average size of itemset       | $S_{is}$ | $ \mathcal{I} $ | 4, 6, & 8               |
| Number of itemsets            | $N_{is}$ | $ \mathcal{L} $ | 800, 1200, 1600, & 2000 |

mean with a few large *itemsets*. Similarly, the size of the user's basket is also clustered around a mean with a few users making lots of purchases.

The IBM dataset generator first creates a list of itemsets and then builds each user's basket from these itemsets. Some of the key parameters that are used by the generator to define the characteristics of the synthetic dataset are shown in Table VIII. The first two parameters,  $n$  and  $m$ , determine the size of the dataset by identifying the number of customers and the number of items (i.e., the  $n \times m$  user-item matrix). The generator creates  $N_{is}$  itemsets whose size is governed by a Poisson distribution having a mean of  $S_{is}$ . The items making up the itemset are chosen randomly with some care taken to ensure that there is some overlap in the different itemsets. After creating the itemsets to be used the dataset is generated by creating a basket of items for each user. The size of the basket follows a Poisson distribution with mean  $S_u$ . Once the size is identified, the basket is filled with itemsets. If an itemset does not fit in the basket then it is added to the basket anyway in half the cases and moved to the next basket in the rest of the cases. To ensure that some itemsets occur more frequently than the rest, each itemset is assigned a weight and the probability of an itemset being selected is governed by that weight. The weights are assigned according to an exponential distribution with mean equal to one. In addition, to create transactions with higher variability, the generator randomly changes some of the items in each itemset as it is inserted into the transaction.

In order to evaluate the effect of the sparsity and the variability in the user's behavior on the overall performance of the various item- and user-based *top-N* recommendation algorithms, we generated 36 different datasets in which we fixed  $n$  and  $m$  but we varied  $S_u$ ,  $S_{is}$ , and  $N_{is}$ . The range of values used to generate the different datasets is shown in the last column of Table VIII. We generated datasets of different sparsity by increasing the average size of the user's basket ( $S_u$ ) while keeping the other two parameters fixed. Specifically, we generated datasets in which each user contained 15, 30, and 45 items on average. We generated datasets with different user's behavioral variability by varying the number of different itemsets ( $N_{is}$ ) and their average size ( $S_{is}$ ). Assuming that  $S_u$  and  $S_{is}$  is kept fixed, by changing the number of different itemsets that can be *packed* to create the various transactions, we can influence how many distinct user-groups exist in the dataset. This is because on average, the generator will combine  $S_u/S_{is}$  itemsets randomly selected from the  $N_{is}$  itemsets to form a particular transaction. By increasing  $N_{is}$  we increase the pool of possible combinations of  $S_u/S_{is}$  itemsets and thus increase the variability (in terms of what items are included in the user's transactions) in the dataset. A somewhat different way of changing the variability of the dataset can be

performed by changing the average size of each itemset. In particular, if we fix  $S_u$  and  $N_{is}$ , then by increasing  $S_{is}$  we decrease the number of possible itemset-combinations that can exist, since now, on average,  $S_u/N_{is}$  itemsets will be included. However, because each such itemset is now larger, this affects the complexity of the purchasing decision represented by that particular itemset.

**6.3.1 Results.** Table IX shows the HR and ARHR achieved by both the user-based scheme and the first- and second-order interpolated item-based schemes that use either the cosine- or the conditional probability-based similarity measure. The results for the user-based scheme were obtained using exactly the same algorithm used in Section 6.2.2, whereas the item-based results were obtained by using  $k = 20$ , and for each dataset we used the  $\alpha$  value that resulted in the highest HR value for the first-order conditional probability-based model. The specific values of  $\alpha$  that were used are shown in the column labeled “ $\alpha$ ”. Also, the second-order models were obtained by using a value of the support threshold of 0.01% for  $S_u = 15$ , 0.1% for  $S_u = 30$ , and 0.5% for  $S_u = 45$ . The number of frequent patterns that were discovered and used in these models is shown in the column labeled “ $F_2$ ”.

The results of Table IX provide a comprehensive comparison of the various algorithms under a wide-range of dataset characteristics. To facilitate the various comparisons we plotted some of the results of Table IX in the graphs shown in Figure 3. Each plot in this graph shows the performance achieved by the various schemes when two out of the three parameters (i.e.,  $S_u$ ,  $N_{is}$  and  $S_{is}$ ) were kept constant. Note that the performance trends in these plots are representative of the performance achieved for different values of the fixed parameters. In the rest of this section, we provide an overview of some of the key trends that can be inferred by comparing and analyzing these results.

First, as illustrated in Figure 3(a,b), the performance (either in terms of HR or ARHR) of the various algorithms decreases as we increase the number of itemsets ( $N_{is}$ ) from 800 to 2000. This performance degradation was expected because as discussed in Section 6.3, by increasing the number of itemsets used to generate the user-item matrix we essentially increase the different types of user-groups that exist in the dataset. Since the overall size of the dataset (in terms of the number of users) remains fixed, the problem of learning accurate *top-N* recommendations for each user becomes harder.

Second, as the sparsity decreases, ( $S_u$  increases from 15 to 45), and  $S_{is}$  and  $N_{is}$  remain fixed, the overall performance of the different schemes decreases (Figure 3(c,d)). We believe that this is also due to the fact that the inherent variability in the dataset also increases, since each user now contains a larger number of itemsets.

Third, the performance of the user-based and second-order item-based algorithms increases as we increase the average size of the itemsets ( $S_{is}$ ) from four to eight, whereas the performance of the first-order item-based schemes tends to decrease (Figure 3(e,f) and Table IX). When  $S_{is}$  is small, the first-order item-based schemes consistently (and in some cases substantially) outperform the user-based scheme. However, as  $S_{is}$  increases, the relative performance gap between these two algorithms shrinks to a point at which the

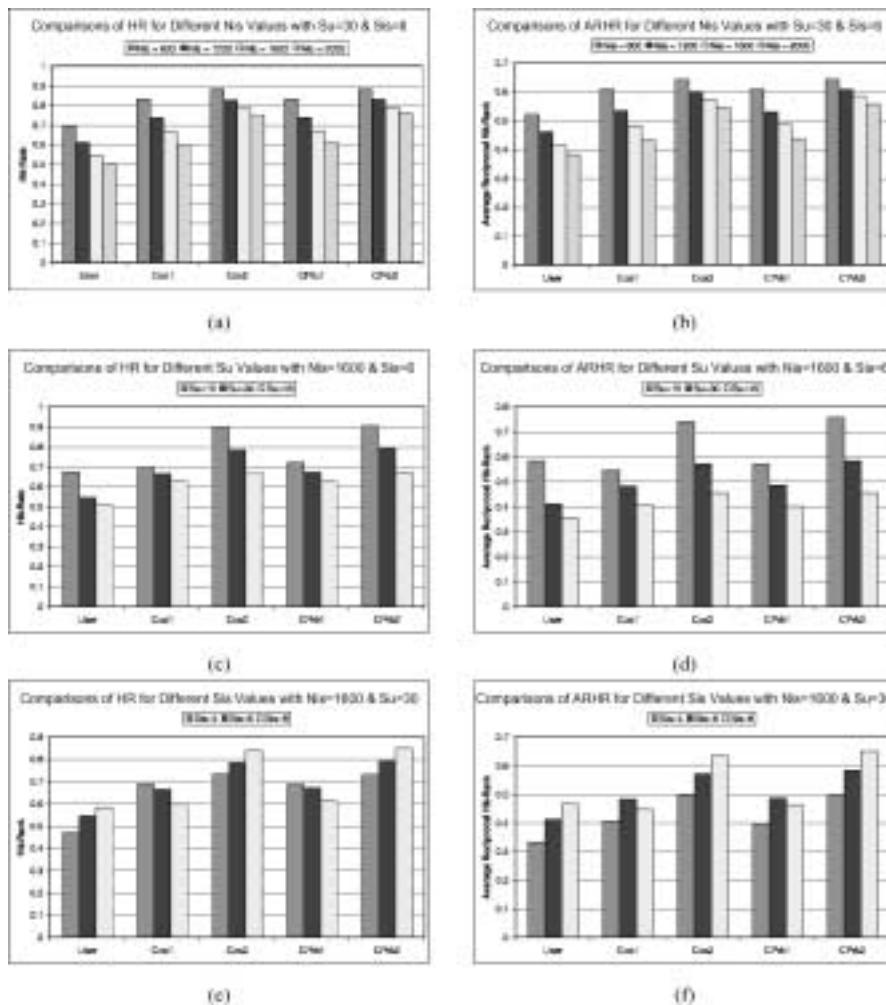
Table IX. The HR and ARHR for Different Values of  $S_u$ ,  $N_{is}$  and  $S_{is}$ 

| Avg. Size of User's Basket ( $S_u$ ) = 15, Sparsity = 1.4 % |          |                            |       |       |       |       |                              |       |       |       |       |        |
|---|----------|----------------------------|-------|-------|-------|-------|------------------------------|-------|-------|-------|-------|--------|
|   |          | Top-10 HR ( $S_{is} = 4$ ) |       |       |       |       | Top-10 ARHR ( $S_{is} = 4$ ) |       |       |       |       |        |
| $N_{is}$  | $\alpha$ | User                       | Cos1  | Cos2  | CPrb1 | CPrb2 | User                         | Cos1  | Cos2  | CPrb1 | CPrb2 | $F_2$  |
| 800   | 0.3      | 0.645                      | 0.871 | 0.908 | 0.877 | 0.913 | 0.511                        | 0.681 | 0.724 | 0.692 | 0.737 | 44100  |
| 1200  | 0.3      | 0.567                      | 0.804 | 0.870 | 0.816 | 0.877 | 0.449                        | 0.609 | 0.676 | 0.629 | 0.693 | 38718  |
| 1600  | 0.3      | 0.527                      | 0.750 | 0.841 | 0.764 | 0.849 | 0.424                        | 0.560 | 0.653 | 0.580 | 0.669 | 33546  |
| 2000  | 0.3      | 0.497                      | 0.700 | 0.816 | 0.715 | 0.824 | 0.405                        | 0.513 | 0.629 | 0.533 | 0.645 | 31274  |
| Top-10 HR ( $S_{is} = 6$ )                                  |          |                            |       |       |       |       |                              |       |       |       |       |        |
| $N_{is}$  | $\alpha$ | User                       | Cos1  | Cos2  | CPrb1 | CPrb2 | User                         | Cos1  | Cos2  | CPrb1 | CPrb2 | $F_2$  |
| 800   | 0.2      | 0.735                      | 0.858 | 0.946 | 0.871 | 0.952 | 0.619                        | 0.689 | 0.786 | 0.710 | 0.804 | 34158  |
| 1200  | 0.2      | 0.686                      | 0.768 | 0.923 | 0.784 | 0.928 | 0.589                        | 0.603 | 0.758 | 0.625 | 0.775 | 29660  |
| 1600  | 0.2      | 0.672                      | 0.700 | 0.897 | 0.720 | 0.903 | 0.583                        | 0.546 | 0.740 | 0.570 | 0.756 | 29318  |
| 2000  | 0.2      | 0.668                      | 0.638 | 0.876 | 0.663 | 0.884 | 0.582                        | 0.494 | 0.722 | 0.515 | 0.740 | 29942  |
| Top-10 HR ( $S_{is} = 8$ )                                  |          |                            |       |       |       |       |                              |       |       |       |       |        |
| $N_{is}$  | $\alpha$ | User                       | Cos1  | Cos2  | CPrb1 | CPrb2 | User                         | Cos1  | Cos2  | CPrb1 | CPrb2 | $F_2$  |
| 800   | 0.2      | 0.811                      | 0.810 | 0.968 | 0.826 | 0.970 | 0.709                        | 0.654 | 0.826 | 0.683 | 0.841 | 29675  |
| 1200  | 0.2      | 0.792                      | 0.703 | 0.949 | 0.727 | 0.953 | 0.703                        | 0.555 | 0.810 | 0.585 | 0.825 | 30456  |
| 1600  | 0.2      | 0.798                      | 0.629 | 0.932 | 0.654 | 0.937 | 0.711                        | 0.498 | 0.795 | 0.525 | 0.811 | 32873  |
| 2000  | 0.2      | 0.804                      | 0.570 | 0.913 | 0.593 | 0.919 | 0.716                        | 0.447 | 0.780 | 0.471 | 0.794 | 35347  |
| Avg. Size of User's Basket ( $S_u$ ) = 30, Sparsity = 2.8 % |          |                            |       |       |       |       |                              |       |       |       |       |        |
|   |          | Top-10 HR ( $S_{is} = 4$ ) |       |       |       |       | Top-10 ARHR ( $S_{is} = 4$ ) |       |       |       |       |        |
| $N_{is}$  | $\alpha$ | User                       | Cos1  | Cos2  | CPrb1 | CPrb2 | User                         | Cos1  | Cos2  | CPrb1 | CPrb2 | $F_2$  |
| 800   | 0.6      | 0.555                      | 0.791 | 0.850 | 0.802 | 0.840 | 0.398                        | 0.427 | 0.589 | 0.491 | 0.577 | 112965 |
| 1200  | 0.6      | 0.505                      | 0.732 | 0.783 | 0.737 | 0.770 | 0.357                        | 0.421 | 0.529 | 0.473 | 0.514 | 108244 |
| 1600  | 0.5      | 0.469                      | 0.686 | 0.732 | 0.685 | 0.730 | 0.332                        | 0.406 | 0.495 | 0.398 | 0.494 | 102016 |
| 2000  | 0.6      | 0.427                      | 0.633 | 0.691 | 0.632 | 0.675 | 0.300                        | 0.393 | 0.462 | 0.414 | 0.447 | 98129  |
| Top-10 HR ( $S_{is} = 6$ )                                  |          |                            |       |       |       |       |                              |       |       |       |       |        |
| $N_{is}$  | $\alpha$ | User                       | Cos1  | Cos2  | CPrb1 | CPrb2 | User                         | Cos1  | Cos2  | CPrb1 | CPrb2 | $F_2$  |
| 800   | 0.5      | 0.690                      | 0.831 | 0.881 | 0.831 | 0.880 | 0.521                        | 0.607 | 0.642 | 0.607 | 0.642 | 107981 |
| 1200  | 0.4      | 0.609                      | 0.734 | 0.824 | 0.736 | 0.829 | 0.459                        | 0.533 | 0.596 | 0.529 | 0.606 | 92674  |
| 1600  | 0.4      | 0.544                      | 0.663 | 0.783 | 0.670 | 0.790 | 0.411                        | 0.482 | 0.571 | 0.485 | 0.583 | 87771  |
| 2000  | 0.4      | 0.497                      | 0.596 | 0.751 | 0.606 | 0.760 | 0.381                        | 0.430 | 0.544 | 0.435 | 0.556 | 85295  |
| Top-10 HR ( $S_{is} = 8$ )                                  |          |                            |       |       |       |       |                              |       |       |       |       |        |
| $N_{is}$  | $\alpha$ | User                       | Cos1  | Cos2  | CPrb1 | CPrb2 | User                         | Cos1  | Cos2  | CPrb1 | CPrb2 | $F_2$  |
| 800   | 0.3      | 0.730                      | 0.780 | 0.907 | 0.794 | 0.915 | 0.570                        | 0.582 | 0.679 | 0.605 | 0.704 | 95648  |
| 1200  | 0.3      | 0.635                      | 0.669 | 0.866 | 0.687 | 0.879 | 0.501                        | 0.500 | 0.651 | 0.517 | 0.675 | 88176  |
| 1600  | 0.4      | 0.582                      | 0.596 | 0.838 | 0.611 | 0.849 | 0.465                        | 0.446 | 0.633 | 0.462 | 0.650 | 82415  |
| 2000  | 0.4      | 0.545                      | 0.533 | 0.816 | 0.549 | 0.828 | 0.442                        | 0.398 | 0.617 | 0.412 | 0.633 | 85814  |
| Avg. Size of User's Basket ( $S_u$ ) = 45, Sparsity = 4.2 % |          |                            |       |       |       |       |                              |       |       |       |       |        |
|   |          | Top-10 HR ( $S_{is} = 4$ ) |       |       |       |       | Top-10 ARHR ( $S_{is} = 4$ ) |       |       |       |       |        |
| $N_{is}$  | $\alpha$ | User                       | Cos1  | Cos2  | CPrb1 | CPrb2 | User                         | Cos1  | Cos2  | CPrb1 | CPrb2 | $F_2$  |
| 800   | 0.8      | 0.464                      | 0.502 | 0.794 | 0.711 | 0.709 | 0.298                        | 0.108 | 0.492 | 0.364 | 0.431 | 12337  |
| 1200  | 0.8      | 0.406                      | 0.567 | 0.714 | 0.655 | 0.618 | 0.262                        | 0.122 | 0.442 | 0.381 | 0.370 | 9334   |
| 1600  | 0.7      | 0.377                      | 0.566 | 0.653 | 0.612 | 0.614 | 0.243                        | 0.150 | 0.403 | 0.294 | 0.375 | 7002   |
| 2000  | 0.7      | 0.350                      | 0.519 | 0.595 | 0.562 | 0.553 | 0.225                        | 0.141 | 0.365 | 0.306 | 0.338 | 6217   |

(Continued)

Table IX. Continued

| Top-10 HR ( $S_{is} = 6$ ) |          |       |       |       |       |       |       | Top-10 ARHR ( $S_{is} = 6$ ) |       |       |       |       |  |  |  |
|----------------------------|----------|-------|-------|-------|-------|-------|-------|------------------------------|-------|-------|-------|-------|--|--|--|
| $N_{is}$                   | $\alpha$ | User  | Cos1  | Cos2  | CPrb1 | CPrb2 | User  | Cos1                         | Cos2  | CPrb1 | CPrb2 | $F_2$ |  |  |  |
| 800                        | 0.6      | 0.580 | 0.775 | 0.812 | 0.778 | 0.789 | 0.410 | 0.469                        | 0.549 | 0.512 | 0.525 | 9661  |  |  |  |
| 1200                       | 0.5      | 0.538 | 0.688 | 0.721 | 0.688 | 0.720 | 0.381 | 0.421                        | 0.484 | 0.417 | 0.484 | 7628  |  |  |  |
| 1600                       | 0.5      | 0.504 | 0.625 | 0.671 | 0.624 | 0.670 | 0.356 | 0.404                        | 0.455 | 0.401 | 0.454 | 6519  |  |  |  |
| 2000                       | 0.5      | 0.468 | 0.562 | 0.627 | 0.561 | 0.626 | 0.331 | 0.369                        | 0.421 | 0.367 | 0.420 | 5367  |  |  |  |
| Top-10 HR ( $S_{is} = 8$ ) |          |       |       |       |       |       |       | Top-10 ARHR ( $S_{is} = 8$ ) |       |       |       |       |  |  |  |
| $N_{is}$                   | $\alpha$ | User  | Cos1  | Cos2  | CPrb1 | CPrb2 | User  | Cos1                         | Cos2  | CPrb1 | CPrb2 | $F_2$ |  |  |  |
| 800                        | 0.4      | 0.672 | 0.757 | 0.817 | 0.759 | 0.829 | 0.495 | 0.533                        | 0.565 | 0.534 | 0.586 | 9568  |  |  |  |
| 1200                       | 0.4      | 0.619 | 0.645 | 0.747 | 0.648 | 0.764 | 0.452 | 0.459                        | 0.515 | 0.454 | 0.536 | 7658  |  |  |  |
| 1600                       | 0.5      | 0.561 | 0.571 | 0.707 | 0.572 | 0.705 | 0.408 | 0.406                        | 0.485 | 0.407 | 0.485 | 6006  |  |  |  |
| 2000                       | 0.5      | 0.507 | 0.509 | 0.676 | 0.509 | 0.675 | 0.369 | 0.362                        | 0.465 | 0.362 | 0.464 | 4652  |  |  |  |

Fig. 3. The HR and ARHR for different values of  $S_u$ ,  $N_{is}$  and  $S_{is}$ .

user-based scheme outperforms the first-order item-based schemes when  $S_{is}$  is eight and  $N_{is}$  is large (e.g.,  $S_u = 15$ ,  $N_{is} = 2000$ , and  $S_{is} = 8$ ). Note that the relative performance advantage of user- versus item-based schemes disappears when we consider the second-order item-based schemes that always and substantially outperform the user-based scheme. The performance gains achieved by the user-based scheme can be explained by the fact that longer itemsets lead to datasets that have lower variability (i.e., each user is described by a small number of itemsets) and they are easier to identify the *correct* user-neighborhood as they will now overlap in a large number of items. However, the reason that the first-order item-based schemes perform worse while the corresponding second-order schemes perform better is somewhat more complicated. By using longer itemsets, the degree of overlap between the different itemsets that are put together to form a transaction increases as well. As a result, for each item  $j$  its similarity distribution to the other  $k$  most similar items becomes less uniform (it tends to have much higher similarities to items that co-occur with  $j$  in the itemset overlaps). Consequently, when these individual item-to-item similarities are combined to form the recommendations, they tend to be biased toward the overlapping items. This problem can be corrected by increasing the model size (i.e., the number of neighbors  $k$  that we store for each item). In fact, we performed a set of experiments in which  $k$  was increased from 20 (used to obtain the results in Table IX) to 50, and this eliminated the degradation in performance of the first-order schemes.

Fourth, comparing the various item-based schemes we can see that, as it was the case with the real datasets, the conditional probability-based approach consistently outperforms the cosine-based approach. Moreover, comparing the values of  $\alpha$  that lead to the best performance of the conditional probability-based approach we notice a similar trend as that described in Section 6.2.1.4, as larger  $\alpha$ -values tend to work better for denser datasets. Finally, the results of Table IX illustrate that for many datasets the second-order item-based models provide a substantial performance improvement. In many cases, the second-order models lead to improvements in the range of 50% to 80%.

## 7. CONCLUSIONS

In this article, we presented and experimentally evaluated a class of model-based *top-N* recommendation algorithms that use item-to-item or itemset-to-item similarities to compute the recommendations. Our results showed that both the conditional probability-based item similarity scheme and higher-order item-based models lead to recommender systems that provide reasonably accurate recommendations that are comparable or better than those provided by traditional user-based CF techniques. Furthermore, the proposed algorithms are substantially faster; allowing real-time recommendations independent of the size of the user-item matrix.

## REFERENCES

- AGGARWAL, C., WOLF, J., WU, K., AND YU, P. 1999. Horting hatches an egg: A new graph-theoretic approach to collaborative filtering. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ACM, New York.

- AGRAWAL, R., IMIELINSKI, T., AND SWAMI, A. 1993. Mining association rules between sets of items in large databases. In *Proceedings of 1993 ACM-SIGMOD International Conference on Management of Data* (Washington, D.C). ACM, New York.
- AGRAWAL, R., MANNILA, H., SRIKANT, R., TOIVONEN, H., AND VERKAMO, A. 1996. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, U. Fayyad, G. Piatetsky-Shapiro, P. Smith, and R. Uthurusamy, Eds. AAAI/MIT Press, Cambridge, Mass., 307–328.
- AGRAWAL, R. AND SRIKANT, R. 1994. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference* (Santiago, Chile.). 487–499.
- BALABANOVIC, M. AND SHOHAM, Y. 1997. FAB: Content-based collaborative recommendation. *Commun. ACM* 40, 3 (Mar.).
- BASU, C., HIRSH, H., AND COHEN, W. 1998. Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the 1998 Workshop on Recommender Systems*. AAAI Press, Reston, Va. 11–15.
- BEEFERMAN, D. AND BERGER, A. 2000. Agglomerative clustering of a search engine query log. In *Proceedings of ACM SIGKDD International Conference*. ACM, New York, 407–415.
- BILLSUS, D. AND PAZZANI, M. J. 1998. Learning collaborative information filters. In *Proceedings of ICML*. 46–53.
- BREESE, J., HECKERMAN, D., AND KADIE, C. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*. 43–52.
- CHAN, P. 1999. A non-invasive learning approach to building web user profiles. In *Proceedings of ACM SIGKDD International Conference*. ACM, New York.
- DELCHER, A. L., HARMON, D., KASIF, S., WHITE, O., AND SALZBERG, S. L. 1998. Improved microbial gene identification with glimmer. *Nucleic Acid Res.* 27, 23, 4436–4641.
- DEMIRIZ, A. 2001. An association mining-based product recommender. In *NFORMS Miami 2001 Annual Meeting Cluster: Data Mining*.
- GOLDBERG, D., NICHOLS, D., OKI, B. M., AND TERRY, D. 1992. Using collaborative filtering to weave an information tapestry. *Commun. ACM* 35, 12, 61–70.
- HECKERMAN, D., CHICKERING, D., MEEK, C., ROUNTHWAITE, R., AND KADIE, C. 2000. Dependency networks for inference, collaborative filtering, and data visualization. *J. Mach. Learn. Res.* 1, 49–75.
- HERLOCKER, J., KONSTAN, J., BORCHERS, A., AND RIEDL, J. 1999. An algorithm framework for performing collaborative filtering. In *Proceedings of SIGIR*. ACM, New York, 77–87.
- HILL, W., STEAD, L., ROSENSTEIN, M., AND FURNAS, G. 1995. Recommending and evaluating choices in a virtual community of use. In *Proceedings of CHI*.
- KARYPIS, G. 2001. Experimental evaluation of item-based top-*n* recommendation algorithms. In *Proceedings of the ACM Conference on Information and Knowledge Management*. ACM, New York.
- KITTS, B., FREED, D., AND VRIEZE, M. 2000. Cross-sell: A fast promotion-tunable customer-item recommendation method based on conditional independent probabilities. In *Proceedings of ACM SIGKDD International Conference*. , ACM, New York, 437–446.
- KONSTAN, J., MILLER, B., MALTZ, D., HERLOCKER, J., GORDON, L., AND RIEDL, J. 1997. GroupLens: Applying collaborative filtering to Usenet news. *Commun. ACM* 40, 3, 77–87.
- LIN, W., ALVAREZ, S., AND RUIZ, C. 2000. Collaborative recommendation via adaptive association rule mining. In *Proceedings of the International Workshop on Web Mining for E-Commerce (WEBKDD'2000)*.
- MCJONES, P. AND DETREVILLE, J. 1997. Each to each programmer's reference manual. Tech. Rep. 1997-023, Systems Research Center. <http://research.compaq.com/SRC/eachmovie/>.
- MOBASHER, B., COOLEY, R., AND SRIVASTAVA, J. 2000. Automatic personalization based on web usage mining. *Commun. ACM* 43, 8, 142–151.
- MOBASHER, B., DAI, H., LUO, T., NAKAGAWA, M., AND WITSHIRE, J. 2000. Discovery of aggregate usage profiles for web personalization. In *Proceedings of the WebKDD Workshop*.
- MOVIELENS 2003. Available at <http://www.grouplens.org/data>.
- RESNICK, P. AND VARIAN, H. R. 1997. Recommender systems. *Commun. ACM* 40, 3, 56–58.
- RESNICK, P., IACOVOU, N., SUCHAK, M., BERGSTROM, P., AND RIEDL, J. 1994. GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of CSCW*.

- SALTON, G. 1989. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, Reading, Mass.
- SARWAR, B., KARYPIS, G., KONSTAN, J., AND RIEDL, J. 2000. Analysis of recommendation algorithms for e-commerce. In *Proceedings of ACM E-Commerce*. ACM, New York.
- SARWAR, B., KARYPIS, G., KONSTAN, J., AND RIEDL, J. 2001. Item-based collaborative filtering recommendation algorithms. In *WWW10*.
- SCHAFER, J., KONSTAN, J., AND RIEDL, J. 1999. Recommender systems in e-commerce. In *Proceedings of ACM E-Commerce*. ACM, New York.
- SENO, M. AND KARYPIS, G. 2001. Lpmiuner: An algorithm for finding frequent itemsets using length-decreasing support constraint. In *Proceedings of the IEEE International Conference on Data Mining*. Also available as a UMN-CS technical report, TR# 01-026.
- SHARDANAND, U. AND MAES, P. 1995. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of the ACM CHI'95 Conference on Human Factors in Computing Systems*. ACM, New York, 210–217.
- TERVEEN, L., HILL, W., AMENTO, B., McDONALD, D., AND CRETER, J. 1997. PHOAKS: A system for sharing recommendations. *Commun. ACM* 40, 3, 59–62.
- UNGAR, L. H. AND FOSTER, D. P. 1998. Clustering methods for collaborative filtering. In *Workshop on Recommendation Systems at the 15th National Conference on Artificial Intelligence*.

Received January 2003; revised June 2003; accepted September 2003

# Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model

Yehuda Koren  
AT&T Labs – Research  
180 Park Ave, Florham Park, NJ 07932  
yehuda@research.att.com

## ABSTRACT

Recommender systems provide users with personalized suggestions for products or services. These systems often rely on Collaborating Filtering (CF), where past transactions are analyzed in order to establish connections between users and products. The two more successful approaches to CF are latent factor models, which directly profile both users and products, and neighborhood models, which analyze similarities between products or users. In this work we introduce some innovations to both approaches. The factor and neighborhood models can now be smoothly merged, thereby building a more accurate combined model. Further accuracy improvements are achieved by extending the models to exploit both explicit and implicit feedback by the users. The methods are tested on the Netflix data. Results are better than those previously published on that dataset. In addition, we suggest a new evaluation metric, which highlights the differences among methods, based on their performance at a top-K recommendation task.

## Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications—*Data Mining*

## General Terms

Algorithms

## Keywords

collaborative filtering, recommender systems

## 1. INTRODUCTION

Modern consumers are inundated with choices. Electronic retailers and content providers offer a huge selection of products, with unprecedented opportunities to meet a variety of special needs and tastes. Matching consumers with most appropriate products is not trivial, yet it is a key in enhancing user satisfaction and loyalty. This emphasizes the prominence of *recommender systems*, which provide personalized recommendations for products that suit a user's taste [1]. Internet leaders like Amazon, Google, Netflix, TiVo and Yahoo are increasingly adopting such recommenders.

Recommender systems are often based on *Collaborative Filtering* (CF) [10], which relies only on past user behavior—e.g., their

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA.

Copyright 2008 ACM 978-1-60558-193-4/08/08 ...\$5.00.

previous transactions or product ratings—and does not require the creation of explicit profiles. Notably, CF techniques require no domain knowledge and avoid the need for extensive data collection. In addition, relying directly on user behavior allows uncovering complex and unexpected patterns that would be difficult or impossible to profile using known data attributes. As a consequence, CF attracted much of attention in the past decade, resulting in significant progress and being adopted by some successful commercial systems, including Amazon [15], TiVo and Netflix.

In order to establish recommendations, CF systems need to compare fundamentally different objects: items against users. There are two primary approaches to facilitate such a comparison, which constitute the two main disciplines of CF: *the neighborhood approach* and *latent factor models*.

Neighborhood methods are centered on computing the relationships between items or, alternatively, between users. An item-oriented approach evaluates the preference of a user to an item based on ratings of similar items by the same user. In a sense, these methods transform users to the item space by viewing them as baskets of rated items. This way, we no longer need to compare users to items, but rather directly relate items to items.

Latent factor models, such as Singular Value Decomposition (SVD), comprise an alternative approach by transforming both items and users to the same latent factor space, thus making them directly comparable. The latent space tries to explain ratings by characterizing both products and users on factors automatically inferred from user feedback. For example, when the products are movies, factors might measure obvious dimensions such as comedy vs. drama, amount of action, or orientation to children; less well defined dimensions such as depth of character development or “quirkiness”; or completely uninterpretable dimensions.

The CF field has enjoyed a surge of interest since October 2006, when the Netflix Prize competition [5] commenced. Netflix released a dataset containing 100 million movie ratings and challenged the research community to develop algorithms that could beat the accuracy of its recommendation system, Cinematch. A lesson that we learnt through this competition is that the neighborhood and latent factor approaches address quite different levels of structure in the data, so none of them is optimal on its own [3].

Neighborhood models are most effective at detecting very localized relationships. They rely on a few significant neighborhood relations, often ignoring the vast majority of ratings by a user. Consequently, these methods are unable to capture the totality of weak signals encompassed in all of a user's ratings. Latent factor models are generally effective at estimating overall structure that relates simultaneously to most or all items. However, these models are poor at detecting strong associations among a small set of closely related items, precisely where neighborhood models do best.

In this work we suggest a combined model that improves predic-

tion accuracy by capitalizing on the advantages of both neighborhood and latent factor approaches. To our best knowledge, this is the first time that a single model has integrated the two approaches. In fact, some past works (e.g., [2, 4]) recognized the utility of combining those approaches. However, they suggested post-processing the factorization results, rather than a unified model where neighborhood and factor information are considered symmetrically.

Another lesson learnt from the Netflix Prize competition is the importance of integrating different forms of user input into the models [3]. Recommender systems rely on different types of input. Most convenient is the high quality *explicit feedback*, which includes explicit input by users regarding their interest in products. For example, Netflix collects star ratings for movies and TiVo users indicate their preferences for TV shows by hitting thumbs-up/down buttons. However, explicit feedback is not always available. Thus, recommenders can infer user preferences from the more abundant *implicit feedback*, which indirectly reflect opinion through observing user behavior [16]. Types of implicit feedback include purchase history, browsing history, search patterns, or even mouse movements. For example, a user that purchased many books by the same author probably likes that author. Our main focus is on cases where explicit feedback is available. Nonetheless, we recognize the importance of implicit feedback, which can illuminate users that did not provide enough explicit feedback. Hence, our models integrate explicit and implicit feedback.

The structure of the rest of the paper is as follows. We start with preliminaries and related work in Sec. 2. Then, we describe a new, more accurate neighborhood model in Sec. 3. The new model is based on an optimization framework that allows smooth integration with latent factor models, and also inclusion of implicit user feedback. Section 4 revisits SVD-based latent factor models while introducing useful extensions. These extensions include a factor model that allows explaining the reasoning behind recommendations. Such explainability is important for practical systems [11, 23] and known to be problematic with latent factor models. The methods introduced in Sec. 3-4 are linked together in Sec. 5, through a model that integrates neighborhood and factor models within a single framework. Relevant experimental results are brought within each section. In addition, we suggest a new methodology to evaluate effectiveness of the models, as described in Sec. 6, with encouraging results.

## 2. PRELIMINARIES

We reserve special indexing letters for distinguishing users from items: for users  $u, v$ , and for items  $i, j$ . A rating  $r_{ui}$  indicates the preference by user  $u$  of item  $i$ , where high values mean stronger preference. For example, values can be integers ranging from 1 (star) indicating no interest to 5 (stars) indicating a strong interest. We distinguish predicted ratings from known ones, by using the notation  $\hat{r}_{ui}$  for the predicted value of  $r_{ui}$ . The  $(u, i)$  pairs for which  $r_{ui}$  is known are stored in the set  $\mathcal{K} = \{(u, i) \mid r_{ui} \text{ is known}\}$ . Usually the vast majority of ratings are unknown. For example, in the Netflix data 99% of the possible ratings are missing. In order to combat overfitting the sparse rating data, models are regularized so estimates are shrunk towards baseline defaults. Regularization is controlled by constants which are denoted as:  $\lambda_1, \lambda_2, \dots$ . Exact values of these constants are determined by cross validation. As they grow, regularization becomes heavier.

### 2.1 Baseline estimates

Typical CF data exhibit large user and item effects – i.e., systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others. It is cus-

tomary to adjust the data by accounting for these effects, which we encapsulate within the *baseline estimates*. Denote by  $\mu$  the overall average rating. A baseline estimate for an unknown rating  $r_{ui}$  is denoted by  $b_{ui}$  and accounts for the user and item effects:

$$b_{ui} = \mu + b_u + b_i \quad (1)$$

The parameters  $b_u$  and  $b_i$  indicate the observed deviations of user  $u$  and item  $i$ , respectively, from the average. For example, suppose that we want a baseline estimate for the rating of the movie *Titanic* by user Joe. Now, say that the average rating over all movies,  $\mu$ , is 3.7 stars. Furthermore, *Titanic* is better than an average movie, so it tends to be rated 0.5 stars above the average. On the other hand, Joe is a critical user, who tends to rate 0.3 stars lower than the average. Thus, the baseline estimate for *Titanic*'s rating by Joe would be 3.9 stars by calculating  $3.7 - 0.3 + 0.5$ . In order to estimate  $b_u$  and  $b_i$  one can solve the least squares problem:

$$\min_{b_*} \sum_{(u,i) \in \mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 (\sum_u b_u^2 + \sum_i b_i^2)$$

Here, the first term  $\sum_{(u,i) \in \mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2$  strives to find  $b_u$ 's and  $b_i$ 's that fit the given ratings. The regularizing term  $-\lambda_1 (\sum_u b_u^2 + \sum_i b_i^2)$  – avoids overfitting by penalizing the magnitudes of the parameters.

### 2.2 Neighborhood models

The most common approach to CF is based on neighborhood models. Its original form, which was shared by virtually all earlier CF systems, is user-oriented; see [12] for a good analysis. Such user-oriented methods estimate unknown ratings based on recorded ratings of like minded users. Later, an analogous item-oriented approach [15, 21] became popular. In those methods, a rating is estimated using known ratings made by the same user on similar items. Better scalability and improved accuracy make the item-oriented approach more favorable in many cases [2, 21, 22]. In addition, item-oriented methods are more amenable to explaining the reasoning behind predictions. This is because users are familiar with items previously preferred by them, but do not know those allegedly like minded users. Thus, our focus is on item-oriented approaches, but parallel techniques can be developed in a user-oriented fashion, by switching the roles of users and items.

Central to most item-oriented approaches is a similarity measure between items. Frequently, it is based on the Pearson correlation coefficient,  $\rho_{ij}$ , which measures the tendency of users to rate items  $i$  and  $j$  similarly. Since many ratings are unknown, it is expected that some items share only a handful of common raters. Computation of the correlation coefficient is based only on the common user support. Accordingly, similarities based on a greater user support are more reliable. An appropriate similarity measure, denoted by  $s_{ij}$ , would be a shrunk correlation coefficient:

$$s_{ij} \stackrel{\text{def}}{=} \frac{n_{ij}}{n_{ij} + \lambda_2} \rho_{ij} \quad (2)$$

The variable  $n_{ij}$  denotes the number of users that rated both  $i$  and  $j$ . A typical value for  $\lambda_2$  is 100. Notice that the literature suggests additional alternatives for a similarity measure [21, 22].

Our goal is to predict  $r_{ui}$  – the unobserved rating by user  $u$  for item  $i$ . Using the similarity measure, we identify the  $k$  items rated by  $u$ , which are most similar to  $i$ . This set of  $k$  neighbors is denoted by  $S^k(i; u)$ . The predicted value of  $r_{ui}$  is taken as a weighted average of the ratings of neighboring items, while adjusting for user and item effects through the baseline estimates:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in S^k(i; u)} s_{ij} (r_{uj} - b_{uj})}{\sum_{j \in S^k(i; u)} s_{ij}} \quad (3)$$

Neighborhood-based methods of this form became very popular because they are intuitive and relatively simple to implement. However, in a recent work [2], we raised a few concerns about such neighborhood schemes. Most notably, these methods are not justified by a formal model. We also questioned the suitability of a similarity measure that isolates the relations between two items, without analyzing the interactions within the full set of neighbors. In addition, the fact that interpolation weights in (3) sum to one forces the method to fully rely on the neighbors even in cases where neighborhood information is absent (i.e., user  $u$  did not rate items similar to  $i$ ), and it would be preferable to rely on baseline estimates.

This led us to propose a more accurate neighborhood model, which overcomes these difficulties. Given a set of neighbors  $S^k(i; u)$  we need to compute *interpolation weights*  $\{\theta_{ij}^u | j \in S^k(i; u)\}$  that enable the best prediction rule of the form:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in S^k(i; u)} \theta_{ij}^u (r_{uj} - b_{uj}) \quad (4)$$

Derivation of the interpolation weights can be done efficiently by estimating all inner products between item ratings; for a full description refer to [2].

### 2.3 Latent factor models

Latent factor models comprise an alternative approach to Collaborative Filtering with the more holistic goal to uncover latent features that explain observed ratings; examples include pLSA [13], neural networks [18], and Latent Dirichlet Allocation [7]. We will focus on models that are induced by Singular Value Decomposition (SVD) on the user-item ratings matrix. Recently, SVD models have gained popularity, thanks to their attractive accuracy and scalability. A typical model associates each user  $u$  with a user-factors vector  $p_u \in \mathbb{R}^f$ , and each item  $i$  with an item-factors vector  $q_i \in \mathbb{R}^f$ . The prediction is done by taking an inner product, i.e.,  $\hat{r}_{ui} = b_{ui} + p_u^T q_i$ . The more involved part is parameter estimation.

In information retrieval it is well established to harness SVD for identifying latent semantic factors [8]. However, applying SVD in the CF domain raises difficulties due to the high portion of missing ratings. Conventional SVD is undefined when knowledge about the matrix is incomplete. Moreover, carelessly addressing only the relatively few known entries is highly prone to overfitting. Earlier works [14, 20] relied on imputation to fill in missing ratings and make the rating matrix dense. However, imputation can be very expensive as it significantly increases the amount of data. In addition, the data may be considerably distorted due to inaccurate imputation. Hence, more recent works [4, 6, 9, 17, 18, 22] suggested modeling directly only the observed ratings, while avoiding overfitting through an adequate regularized model, such as:

$$\min_{p^*, q^*, b^*} \sum_{(u, i) \in \mathcal{K}} (r_{ui} - \mu - b_u - b_i - p_u^T q_i)^2 + \lambda_3 (\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2) \quad (5)$$

A simple gradient descent technique was applied successfully to solving (5).

Paterek [17] suggested the related NSVD model, which avoids explicitly parameterizing each user, but rather models users based on the items that they rated. This way, each item  $i$  is associated with two factor vectors  $q_i$  and  $x_i$ . The representation of a user  $u$  is through the sum:  $(\sum_{j \in R(u)} x_j) / \sqrt{|R(u)|}$ , so  $r_{ui}$  is predicted as:  $b_{ui} + q_i^T (\sum_{j \in R(u)} x_j) / \sqrt{|R(u)|}$ . Here,  $R(u)$  is the set of items rated by user  $u$ . Later in this work, we adapt Paterek's idea with some extensions.

### 2.4 The Netflix data

We evaluated our algorithms on the Netflix data of more than 100 million movie ratings performed by anonymous Netflix customers [5]. We are not aware of any publicly available CF dataset that is close to the scope and quality of this dataset. To maintain compatibility with results published by others, we adopted some standards that were set by Netflix, as follows. First, quality of the results is usually measured by their root mean squared error (RMSE):  $\sqrt{\sum_{(u, i) \in TestSet} (r_{ui} - \hat{r}_{ui})^2 / |TestSet|}$ . In addition, we report results on a test set provided by Netflix (also known as the Quiz set), which contains over 1.4 million recent ratings. Netflix compiled another 1.4 million recent ratings into a validation set, known as the Probe set, which we employ in Section 6. The two sets contain many more ratings by users that do not rate much and are harder to predict. In a way, they represent real requirements from a CF system, which needs to predict new ratings from older ones, and to equally address all users, not only the heavy raters.

The Netflix data is part of the ongoing Netflix Prize competition, where the benchmark is Netflix's proprietary system, Cinematch, which achieved a RMSE of 0.9514 on the test set. The grand prize will be awarded to a team that manages to drive this RMSE below 0.8563 (10% improvement). Results reported in this work lower the RMSE on the test set to levels around 0.887, which is better than previously published results on this dataset.

### 2.5 Implicit feedback

As stated earlier, an important goal of this work is devising models that allow integration of explicit and implicit user feedback. For a dataset such as the Netflix data, the most natural choice for implicit feedback would be movie rental history, which tells us about user preferences without requiring them to explicitly provide their ratings. However, such data is not available to us. Nonetheless, a less obvious kind of implicit data does exist within the Netflix dataset. The dataset does not only tell us the rating values, but also *which* movies users rate, regardless of *how* they rated these movies. In other words, a user implicitly tells us about her preferences by choosing to voice her opinion and vote a (high or low) rating. This reduces the ratings matrix into a binary matrix, where "1" stands for "rated", and "0" for "not rated". Admittedly, this binary data is not as vast and independent as other sources of implicit feedback could be. Nonetheless, we have found that incorporating this kind of implicit data – which inherently exist in every rating based recommender system – significantly improves prediction accuracy. Some prior techniques, such as Conditional RBMs [18], also capitalized on the same binary view of the data.

The models that we suggest are not limited to a certain kind of implicit data. To keep generality, each user  $u$  is associated with two sets of items, one is denoted by  $R(u)$ , and contains all the items for which ratings by  $u$  are available. The other one, denoted by  $N(u)$ , contains all items for which  $u$  provided an implicit preference.

## 3. A NEIGHBORHOOD MODEL

In this section we introduce a new neighborhood model, which allows an efficient global optimization scheme. The model offers improved accuracy and is able to integrate implicit user feedback. We will gradually construct the various components of the model, through an ongoing refinement of our formulations.

Previous models were centered around *user-specific* interpolation weights –  $\theta_{ij}^u$  in (4) or  $s_{ij} / \sum_{j \in S^k(i; u)} s_{ij}$  in (3) – relating item  $i$  to the items in a user-specific neighborhood  $S^k(i; u)$ . In

order to facilitate global optimization, we would like to abandon such user-specific weights in favor of global weights independent of a specific user. The weight from  $j$  to  $i$  is denoted by  $w_{ij}$  and will be learnt from the data through optimization. An initial sketch of the model describes each rating  $r_{ui}$  by the equation:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in R(u)} (r_{uj} - b_{uj})w_{ij} \quad (6)$$

For now, (6) does not look very different from (4), besides our claim that the  $w_{ij}$ 's are not user specific. Another difference, which will be discussed shortly, is that here we sum over all items rated by  $u$ , unlike (4) that sums over members of  $S^k(i; u)$ .

Let us consider the interpretation of the weights. Usually the weights in a neighborhood model represent interpolation coefficients relating unknown ratings to existing ones. Here, it is useful to adopt a different viewpoint, where weights represent offsets to baseline estimates. Now, the residuals,  $r_{uj} - b_{uj}$ , are viewed as the coefficients multiplying those offsets. For two related items  $i$  and  $j$ , we expect  $w_{ij}$  to be high. Thus, whenever a user  $u$  rated  $j$  higher than expected ( $r_{uj} - b_{uj}$  is high), we would like to increase our estimate for  $u$ 's rating of  $i$  by adding  $(r_{uj} - b_{uj})w_{ij}$  to the baseline estimate. Likewise, our estimate will not deviate much from the baseline by an item  $j$  that  $u$  rated just as expected ( $r_{uj} - b_{uj}$  is around zero), or by an item  $j$  that is not known to be predictive on  $i$  ( $w_{ij}$  is close to zero). This viewpoint suggests several enhancements to (6). First, we can use implicit feedback, which provide an alternative way to learn user preferences. To this end, we add another set of weights, and rewrite (6) as:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in R(u)} (r_{uj} - b_{uj})w_{ij} + \sum_{j \in N(u)} c_{ij} \quad (7)$$

Much like the  $w_{ij}$ 's, the  $c_{ij}$ 's are offsets added to baseline estimates. For two items  $i$  and  $j$ , an implicit preference by  $u$  to  $j$  lead us to modify our estimate of  $r_{ui}$  by  $c_{ij}$ , which is expected to be high if  $j$  is predictive on  $i$ .<sup>1</sup>

Viewing the weights as global offsets, rather than as user-specific interpolation coefficients, emphasizes the influence of missing ratings. In other words, a user's opinion is formed not only by what he rated, but also by what he did not rate. For example, suppose that a movie ratings dataset shows that users that rate "Lord of the Rings 3" high also gave high ratings to "Lord of the Rings 1–2". This will establish high weights from "Lord of the Rings 1–2" to "Lord of the Rings 3". Now, if a user did not rate "Lord of the Rings 1–2" at all, his predicted rating for "Lord of the Rings 3" will be penalized, as some necessary weights cannot be added to the sum.

For prior models ((3),(4)) that interpolated  $r_{ui} - b_{ui}$  from  $\{r_{uj} - b_{uj} | j \in S^k(i; u)\}$ , it was necessary to maintain compatibility between the  $b_{ui}$  values and the  $b_{uj}$  values. However, here we do not use interpolation, so we can decouple the definitions of  $b_{ui}$  and  $b_{uj}$ . Accordingly, a more general prediction rule would be:  $\hat{r}_{ui} = \tilde{b}_{ui} + \sum_{j \in R(u)} (r_{uj} - b_{uj})w_{ij} + \sum_{j \in N(u)} c_{ij}$ . Here,  $\tilde{b}_{ui}$  represents predictions of  $r_{ui}$  by other methods such as a latent factor model. We elaborate more on this in Section 5. For now, we suggest the following rule that was found to work well:

$$\hat{r}_{ui} = \mu + b_u + b_i + \sum_{j \in R(u)} (r_{uj} - b_{uj})w_{ij} + \sum_{j \in N(u)} c_{ij} \quad (8)$$

Importantly, the  $b_{uj}$ 's remain constants, which are derived as explained in Sec. 2.1. However, the  $b_u$ 's and  $b_i$ 's become parameters, which are optimized much like the  $w_{ij}$ 's and  $c_{ij}$ 's.

<sup>1</sup>In many cases it would be reasonable to attach significance weights to implicit feedback. This requires a modification to our formula which, for simplicity, will not be considered here.

A characteristic of the current scheme is that it encourages greater deviations from baseline estimates for users that provided many ratings (high  $|R(u)|$ ) or plenty of implicit feedback (high  $|N(u)|$ ). In general, this is a good practice for recommender systems. We would like to take more risk with well modeled users that provided much input. For such users we are willing to predict quirkier and less common recommendations. On the other hand, we are less certain about the modeling of users that provided only a little input, in which case we would like to stay with safe estimates close to the baseline values. However, our experience shows that the current model somewhat overemphasizes the dichotomy between heavy raters and those that rarely rate. Better results were obtained when we moderated this behavior, replacing the prediction rule with:

$$\begin{aligned} \hat{r}_{ui} = & \mu + b_u + b_i + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} (r_{uj} - b_{uj})w_{ij} \\ & + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} c_{ij} \end{aligned} \quad (9)$$

Complexity of the model can be reduced by pruning parameters corresponding to unlikely item-item relations. Let us denote by  $S^k(i)$  the set of  $k$  items most similar to  $i$ , as determined by the similarity measure  $s_{ij}$ . Additionally, we use  $R^k(i; u) \stackrel{\text{def}}{=} R(u) \cap S^k(i)$  and  $N^k(i; u) \stackrel{\text{def}}{=} N(u) \cap S^k(i)$ .<sup>2</sup> Now, when predicting  $r_{ui}$  according to (9), it is expected that the most influential weights will be associated with items similar to  $i$ . Hence, we replace (9) with:

$$\begin{aligned} \hat{r}_{ui} = & \mu + b_u + b_i + |R^k(i; u)|^{-\frac{1}{2}} \sum_{j \in R^k(i; u)} (r_{uj} - b_{uj})w_{ij} \\ & + |N^k(i; u)|^{-\frac{1}{2}} \sum_{j \in N^k(i; u)} c_{ij} \end{aligned} \quad (10)$$

When  $k = \infty$ , rule (10) coincides with (9). However, for other values of  $k$  it offers the potential to significantly reduce the number of variables involved.

This is our final prediction rule, which allows fast online prediction. More computational work is needed at a pre-processing stage where parameters are estimated. A major design goal of the new neighborhood model was facilitating an efficient global optimization procedure, which prior neighborhood models lacked. Thus, model parameters are learnt by solving the regularized least squares problem associated with (10):

$$\begin{aligned} \min_{b_*, w_*, c_*} \sum_{(u, i) \in \mathcal{K}} & \left( r_{ui} - \mu - b_u - b_i - |N^k(i; u)|^{-\frac{1}{2}} \sum_{j \in N^k(i; u)} c_{ij} \right. \\ & \left. - |R^k(i; u)|^{-\frac{1}{2}} \sum_{j \in R^k(i; u)} (r_{uj} - b_{uj})w_{ij} \right)^2 \\ & + \lambda_4 \left( b_u^2 + b_i^2 + \sum_{j \in R^k(i; u)} w_{ij}^2 + \sum_{j \in N^k(i; u)} c_{ij}^2 \right) \end{aligned} \quad (11)$$

An optimal solution of this convex problem can be obtained by

<sup>2</sup>Notational clarification: With other neighborhood models it was beneficial to use  $S^k(i; u)$ , which denotes the  $k$  items most similar to  $i$  among those rated by  $u$ . Hence, if  $u$  rated at least  $k$  items, we will always have  $|S^k(i; u)| = k$ , regardless of how similar those items are to  $i$ . However,  $|R^k(i; u)|$  is typically smaller than  $k$ , as some of those items most similar to  $i$  were not rated by  $u$ .

least square solvers, which are part of standard linear algebra packages. However, we have found that the following simple gradient descent solver works much faster. Let us denote the prediction error,  $r_{ui} - \hat{r}_{ui}$ , by  $e_{ui}$ . We loop through all known ratings in  $\mathcal{K}$ . For a given training case  $r_{ui}$ , we modify the parameters by moving in the opposite direction of the gradient, yielding:

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda_4 \cdot b_i)$
- $\forall j \in R^k(i; u) :$   
 $w_{ij} \leftarrow w_{ij} + \gamma \cdot \left( |R^k(i; u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_4 \cdot w_{ij} \right)$
- $\forall j \in N^k(i; u) :$   
 $c_{ij} \leftarrow c_{ij} + \gamma \cdot \left( |N^k(i; u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_4 \cdot c_{ij} \right)$

The meta-parameters  $\gamma$  (step size) and  $\lambda_4$  are determined by cross-validation. We used  $\gamma = 0.005$  and  $\lambda_4 = 0.002$  for the Netflix data. A typical number of iterations throughout the training data is 15. Another important parameter is  $k$ , which controls the neighborhood size. Our experience shows that increasing  $k$  always benefits the accuracy of the results on the test set. Hence, the choice of  $k$  should reflect a tradeoff between prediction accuracy and computational cost.

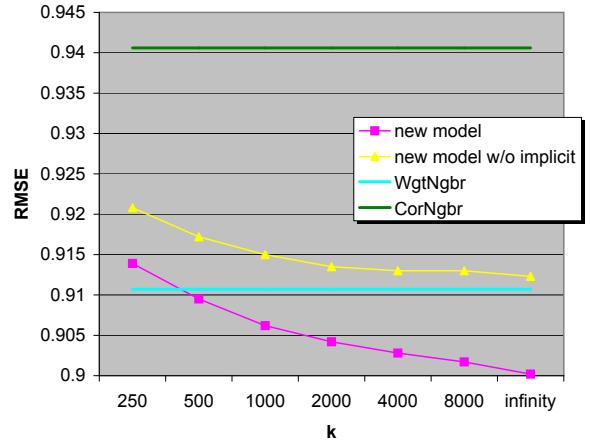
Experimental results on the Netflix data with the new neighborhood model are presented in Fig. 1. We studied the model under different values of parameter  $k$ . The pink curve shows that accuracy monotonically improves with rising  $k$  values, as root mean squared error (RMSE) falls from 0.9139 for  $k = 250$  to 0.9002 for  $k = \infty$ . (Notice that since the Netflix data contains 17,770 movies,  $k = \infty$  is equivalent to  $k = 17,770$ , where all item-item relations are explored.) We repeated the experiments without using the implicit feedback, that is, dropping the  $c_{ij}$  parameters from our model. The results depicted by the yellow curve show a significant decline in estimation accuracy, which widens as  $k$  grows. This demonstrates the value of incorporating implicit feedback into the model.

For comparison we provide the results of two previous neighborhood models. First is a correlation-based neighborhood model (following (3)), which is the most popular CF method in the literature. We denote this model as CorNgbr. Second is a newer model [2] that follows (4), which will be denoted as WgtNgbr. For both these two models, we tried to pick optimal parameters and neighborhood sizes, which were 20 for CorNgbr, and 50 for WgtNgbr. The results are depicted by the green and cyan lines. Notice that the  $k$  value (the  $x$ -axis) is irrelevant to these models, as their different notion of neighborhood makes neighborhood sizes incompatible. It is clear that the popular CorNgbr method is noticeably less accurate than the other neighborhood models, though its 0.9406 RMSE is still better than the published Netflix’s Cinematch RMSE of 0.9514. On the opposite side, our new model delivers more accurate results even when compared with WgtNgbr, as long as the value of  $k$  is at least 500.

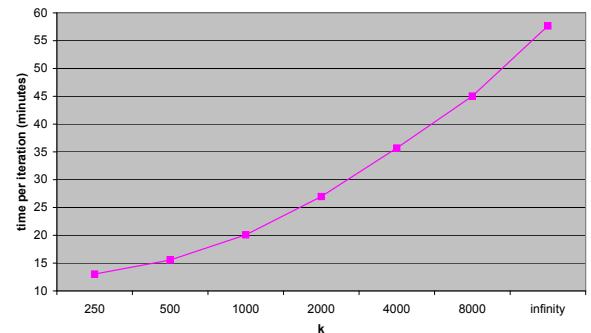
Finally, let us consider running time. Previous neighborhood models require very light pre-processing, though, WgtNgbr [2] requires solving a small system of equations for each provided prediction. The new model does involve pre-processing where parameters are estimated. However, online prediction is immediate by following rule (10). Pre-processing time grows with the value of  $k$ . Typical running times per iteration on the Netflix data, as measured on a single processor 3.4GHz Pentium 4 PC, are shown in Fig. 2.

## 4. LATENT FACTOR MODELS REVISITED

As mentioned in Sec. 2.3, a popular approach to latent factor models is induced by an SVD-like lower rank decomposition of the



**Figure 1: Comparison of neighborhood-based models.** We measure the accuracy of the new model with and without implicit feedback. Accuracy is measured by RMSE on the Netflix test set, so lower values indicate better performance. RMSE is shown as a function of varying values of  $k$ , which dictates the neighborhood size. For reference, we present the accuracy of two prior models as two horizontal lines: the green line represents a popular method using Pearson correlations, and the cyan line represents a more recent neighborhood model.



**Figure 2: Running times (minutes) per iteration of the neighborhood model, as a function of the parameter  $k$ .**

ratings matrix. Each user  $u$  is associated with a user-factors vector  $p_u \in \mathbb{R}^f$ , and each item  $i$  with an item-factors vector  $q_i \in \mathbb{R}^f$ . Prediction is done by the rule:

$$\hat{r}_{ui} = b_{ui} + p_u^T q_i \quad (12)$$

Parameters are estimated by minimizing the associated squared error function (5). Funk [9] popularized gradient descent optimization, which was successfully practiced by many others [17, 18, 22]. Henceforth, we will dub this basic model “SVD”. We would like to extend the model by considering also implicit information. Following Paterek [17] and our work in the previous section, we suggest the following prediction rule:

$$\begin{aligned} \hat{r}_{ui} = b_{ui} + q_i^T & \left( |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} (r_{uj} - b_{uj}) x_j \right. \\ & \left. + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right) \end{aligned} \quad (13)$$

Here, each item  $i$  is associated with three factor vectors  $q_i, x_i, y_i \in \mathbb{R}^f$ . On the other hand, instead of providing an explicit parameterization for users, we represent users through the items that they prefer. Thus, the previous user factor  $p_i$  was replaced by the sum  $|\mathcal{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}(u)} (r_{uj} - b_{uj})x_j + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j$ . This new model, which will be henceforth named ‘‘Asymmetric-SVD’’, offers several benefits:

1. *Fewer parameters.* Typically the number of users is much larger than the number of products. Thus, exchanging user-parameters with item-parameters lowers the complexity of the model.
2. *New users.* Since Asymmetric-SVD does not parameterize users, we can handle new users as soon as they provide feedback to the system, without needing to re-train the model and estimate new parameters. Similarly, we can immediately exploit new ratings for updating user preferences. Notice that for new items we do have to learn new parameters. Interestingly, this asymmetry between users and items meshes well with common practices: systems need to provide immediate recommendations to new users who expect quality service. On the other hand, it is reasonable to require a waiting period before recommending items new to the system. As a side remark, it is worth mentioning that item-oriented neighborhood models exhibit the same desired asymmetry.
3. *Explainability.* Users expect a system to give a reason for its predictions, rather than facing ‘‘black box’’ recommendations. This not only enriches the user experience, but also encourages users to interact with the system, fix wrong impressions and improve long-term accuracy. In fact, the importance of explaining automated recommendations is widely recognized [11, 23]. Latent factor models such as SVD face real difficulties to explain predictions. After all, a key to these models is abstracting users via an intermediate layer of user factors. This intermediate layer separates the computed predictions from past user actions and complicates explanations. However, the new Asymmetric-SVD model does not employ any level of abstraction on the users side. Hence, predictions are a direct function of past users’ feedback. Such a framework allows identifying which of the past user actions are most influential on the computed prediction, thereby explaining predictions by most relevant actions. Once again, we would like to mention that item-oriented neighborhood models enjoy the same benefit.
4. *Efficient integration of implicit feedback.* Prediction accuracy is improved by considering also implicit feedback, which provides an additional indication of user preferences. Obviously, implicit feedback becomes increasingly important for users that provide much more implicit feedback than explicit one. Accordingly, in rule (13) the implicit perspective becomes more dominant as  $|\mathcal{N}(u)|$  increases and we have much implicit feedback. On the other hand, the explicit perspective becomes more significant when  $|\mathcal{R}(u)|$  is growing and we have many explicit observations. Typically, a single explicit input would be more valuable than a single implicit input. The right conversion ratio, which represents how many implicit inputs are as significant as a single explicit input, is automatically learnt from the data by setting the relative values of the  $x_j$  and  $y_j$  parameters.

As usual, we learn the values of involved parameters by minimizing the regularized squared error function associated with (13):

$$\begin{aligned} & \min_{q_*, x_*, y_*, b_*} \sum_{(u,i) \in \mathcal{K}} \left( r_{ui} - \mu - b_u - b_i \right. \\ & \quad \left. - q_i^T \left( |\mathcal{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}(u)} (r_{uj} - b_{uj})x_j + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j \right) \right)^2 \\ & \quad + \lambda_5 \left( b_u^2 + b_i^2 + \|q_i\|^2 + \sum_{j \in \mathcal{R}(u)} \|x_j\|^2 + \sum_{j \in \mathcal{N}(u)} \|y_j\|^2 \right) \end{aligned} \quad (14)$$

We employ a simple gradient descent scheme to solve the system. On the Netflix data we used 30 iterations, with step size of 0.002 and  $\lambda_5 = 0.04$ .

An important question is whether we need to give up some predictive accuracy in order to enjoy those aforementioned benefits of Asymmetric-SVD. We evaluated this on the Netflix data. As shown in Table 1, prediction quality of Asymmetric-SVD is actually slightly better than SVD. The improvement is likely thanks to accounting for implicit feedback. This means that one can enjoy the benefits that Asymmetric-SVD offers, without sacrificing prediction accuracy. As mentioned earlier, we do not really have much independent implicit feedback for the Netflix dataset. Thus, we expect that for real life systems with access to better types of implicit feedback (such as rental/purchase history), the new Asymmetric-SVD model would lead to even further improvements. Nevertheless, this still has to be demonstrated experimentally.

In fact, as far as integration of implicit feedback is concerned, we could get more accurate results by a more direct modification of (12), leading to the following model:

$$\hat{r}_{ui} = b_{ui} + q_i^T \left( p_u + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j \right) \quad (15)$$

Now, a user  $u$  is modeled as  $p_u + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j$ . We use a free user-factors vector,  $p_u$ , much like in (12), which is learnt from the given explicit ratings. This vector is complemented by the sum  $|\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j$ , which represents the perspective of implicit feedback. We dub this model ‘‘SVD++’’. Similar models were discussed recently [3, 19]. Model parameters are learnt by minimizing the associated squared error function through gradient descent. SVD++ does not offer the previously mentioned benefits of having less parameters, conveniently handling new users and readily explainable results. This is because we do abstract each user with a factors vector. However, as Table 1 indicates, SVD++ is clearly advantageous in terms of prediction accuracy. Actually, to our best knowledge, its results are more accurate than all previously published methods on the Netflix data. Nonetheless, in the next section we will describe an integrated model, which offers further accuracy gains.

## 5. AN INTEGRATED MODEL

The new neighborhood model of Sec. 3 is based on a formal model, whose parameters are learnt by solving a least squares problem. An advantage of this approach is allowing easy integration with other methods that are based on similarly structured global cost functions. As explained in Sec. 1, latent factor models and neighborhood models nicely complement each other. Accordingly, in this section we will integrate the neighborhood model with our most accurate factor model – SVD++. A combined model will sum

| Model          | 50 factors | 100 factors | 200 factors |
|----------------|------------|-------------|-------------|
| SVD            | 0.9046     | 0.9025      | 0.9009      |
| Asymmetric-SVD | 0.9037     | 0.9013      | 0.9000      |
| SVD++          | 0.8952     | 0.8924      | 0.8911      |

**Table 1: Comparison of SVD-based models: prediction accuracy is measured by RMSE on the Netflix test set for varying number of factors ( $f$ ). Asymmetric-SVD offers practical advantages over the known SVD model, while slightly improving accuracy. Best accuracy is achieved by SVD++, which directly incorporates implicit feedback into the SVD model.**

the predictions of (10) and (15), thereby allowing neighborhood and factor models to enrich each other, as follows:

$$\begin{aligned} \hat{r}_{ui} = & \mu + b_u + b_i + q_i^T \left( p_u + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j \right) \\ & + |\mathcal{R}^k(i; u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{R}^k(i; u)} (r_{uj} - b_{uj}) w_{ij} + |\mathcal{N}^k(i; u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}^k(i; u)} c_{ij} \end{aligned} \quad (16)$$

In a sense, rule (16) provides a 3-tier model for recommendations. The first tier,  $\mu + b_u + b_i$ , describes general properties of the item and the user, without accounting for any involved interactions. For example, this tier could argue that “The Sixth Sense” movie is known to be good, and that the rating scale of our user, Joe, tends to be just on average. The next tier,  $q_i^T \left( p_u + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j \right)$ , provides the interaction between the user profile and the item profile. In our example, it may find that “The Sixth Sense” and Joe are rated high on the Psychological Thrillers scale. The final “neighborhood tier” contributes fine grained adjustments that are hard to profile, such as the fact that Joe rated low the related movie “Signs”.

Model parameters are determined by minimizing the associated regularized squared error function through gradient descent. Recall that  $e_{ui} \stackrel{\text{def}}{=} r_{ui} - \hat{r}_{ui}$ . We loop over all known ratings in  $\mathcal{K}$ . For a given training case  $r_{ui}$ , we modify the parameters by moving in the opposite direction of the gradient, yielding:

- $b_u \leftarrow b_u + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_u)$
- $b_i \leftarrow b_i + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_i)$
- $q_i \leftarrow q_i + \gamma_2 \cdot (e_{ui} \cdot (p_u + |\mathcal{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathcal{N}(u)} y_j) - \lambda_7 \cdot q_i)$
- $p_u \leftarrow p_u + \gamma_2 \cdot (e_{ui} \cdot q_i - \lambda_7 \cdot p_u)$
- $\forall j \in \mathcal{N}(u) :$   
 $y_j \leftarrow y_j + \gamma_2 \cdot (e_{ui} \cdot |\mathcal{N}(u)|^{-\frac{1}{2}} \cdot q_i - \lambda_7 \cdot y_j)$
- $\forall j \in \mathcal{R}^k(i; u) :$   
 $w_{ij} \leftarrow w_{ij} + \gamma_3 \cdot \left( |\mathcal{R}^k(i; u)|^{-\frac{1}{2}} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_8 \cdot w_{ij} \right)$
- $\forall j \in \mathcal{N}^k(i; u) :$   
 $c_{ij} \leftarrow c_{ij} + \gamma_3 \cdot \left( |\mathcal{N}^k(i; u)|^{-\frac{1}{2}} \cdot e_{ui} - \lambda_8 \cdot c_{ij} \right)$

When evaluating the method on the Netflix data, we used the following values for the meta parameters:  $\gamma_1 = \gamma_2 = 0.007$ ,  $\gamma_3 = 0.001$ ,  $\lambda_6 = 0.005$ ,  $\lambda_7 = \lambda_8 = 0.015$ . It is beneficial to decrease step sizes (the  $\gamma$ 's) by a factor of 0.9 after each iteration. The neighborhood size,  $k$ , was set to 300. Unlike the pure neighborhood model (10), here there is no benefit in increasing  $k$ , as adding neighbors covers more global information, which the latent factors already capture adequately. The iterative process runs for around

|                | 50 factors | 100 factors | 200 factors |
|----------------|------------|-------------|-------------|
| RMSE           | 0.8877     | 0.8870      | 0.8868      |
| time/iteration | 17min      | 20min       | 25min       |

**Table 2: Performance of the integrated model. Prediction accuracy is improved by combining the complementing neighborhood and latent factor models. Increasing the number of factors contributes to accuracy, but also adds to running time.**

30 iterations till convergence. Table 2 summarizes the performance over the Netflix dataset for different number of factors. Once again, we report running times on a Pentium 4 PC for processing the 100 million ratings Netflix data. By coupling neighborhood and latent factor models together, and recovering signal from implicit feedback, accuracy of results is improved beyond other methods.

Recall that unlike SVD++, both the neighborhood model and Asymmetric-SVD allow a direct explanation of their recommendations, and do not require re-training the model for handling new users. Hence, when explainability is preferred over accuracy, one can follow very similar steps to integrate Asymmetric-SVD with the neighborhood model, thereby improving accuracy of the individual models while still maintaining the ability to reason about recommendations to end users.

## 6. EVALUATION THROUGH A TOP-K RECOMMENDER

So far, we have followed a common practice with the Netflix dataset to evaluate prediction accuracy by the RMSE measure. Achievable RMSE values on the Netflix test data lie in a quite narrow range. A simple prediction rule, which estimates  $r_{ui}$  as the mean rating of movie  $i$ , will result in RMSE=1.053. Notice that this rule represents a sensible “best sellers list” approach, where the same recommendation applies to all users. By applying personalization, more accurate predictions are obtained. This way, Netflix Cinematch system could achieve a RMSE of 0.9514. In this paper, we suggested methods that lower the RMSE to 0.8870. In fact, by blending several solutions, we could reach a RMSE of 0.8645. Nonetheless, none of the 3,400 teams actively involved in the Netflix Prize competition could reach, as of 20 months into the competition, lower RMSE levels, despite the big incentive of winning a \$1M Grand Prize. Thus, the range of attainable RMSEs is seemingly compressed, with less than 20% gap between a naive non-personalized approach and the best known CF results. Successful improvements of recommendation quality depend on achieving the elusive goal of enhancing users' satisfaction. Thus, a crucial question is: what effect on user experience should we expect by lowering the RMSE by, say, 10%? For example, is it possible, that a solution with a slightly better RMSE will lead to completely different and better recommendations? This is central to justifying research on accuracy improvements in recommender systems. We would like to shed some light on the issue, by examining the effect of lowered RMSE on a practical situation.

A common case facing recommender systems is providing “top K recommendations”. That is, the system needs to suggest the top K products to a user. For example, recommending the user a few specific movies which are supposed to be most appealing to him. We would like to investigate the effect of lowering the RMSE on the quality of top K recommendations. Somewhat surprisingly, the Netflix dataset can be used to evaluate this.

Recall that in addition to the test set, Netflix also provided a validation set for which the true ratings are published. We used all 5-star ratings from the validation set as a proxy for movies that

interest users.<sup>3</sup> Our goal is to find the relative place of these “interesting movies” within the total order of movies sorted by predicted ratings for a specific user. To this end, for each such movie  $i$ , rated 5-stars by user  $u$ , we select 1000 additional random movies and predict the ratings by  $u$  for  $i$  and for the other 1000 movies. Finally, we order the 1001 movies based on their predicted rating, in a decreasing order. This simulates a situation where the system needs to recommend movies out of 1001 available ones. Thus, those movies with the highest predictions will be recommended to user  $u$ . Notice that the 1000 movies are random, some of which may be of interest to user  $u$ , but most of them are probably of no interest to  $u$ . Hence, the best hoped result is that  $i$  (for which we know  $u$  gave the highest rating of 5) will precede the rest 1000 random movies, thereby improving the appeal of a top-K recommender. There are 1001 different possible ranks for  $i$ , ranging from the best case where none (0%) of the random movies appears before  $i$ , to the worst case where all (100%) of the random movies appear before  $i$  in the sorted order. Overall, the validation set contains 384,573 5-star ratings. For each of them (separately) we draw 1000 random movies, predict associated ratings, and derive a ranking between 0% to 100%. Then, the distribution of the 384,573 ranks is analyzed. (Remark: since the number 1000 is arbitrary, reported results are in percentiles (0%–100%), rather than in absolute ranks (0–1000).)

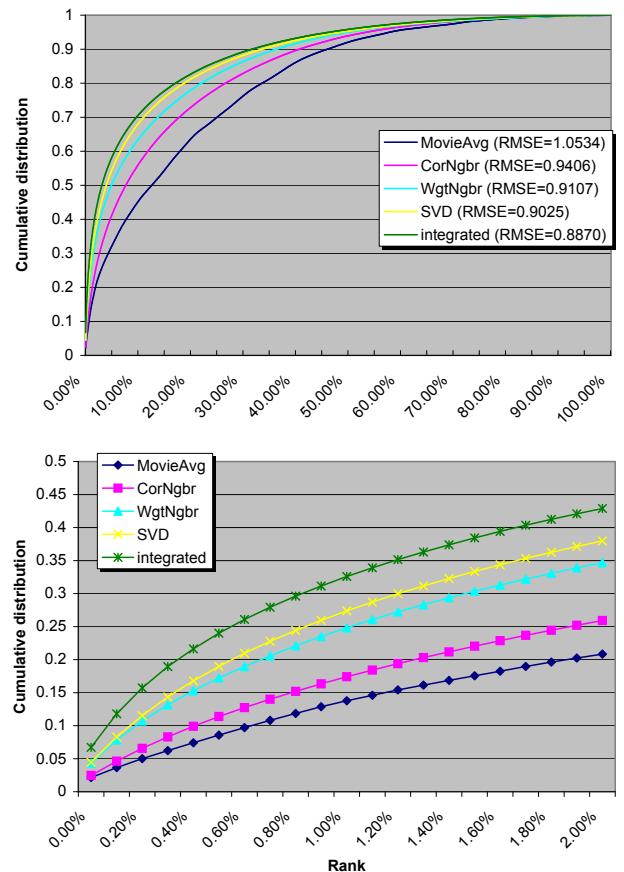
We used this methodology to evaluate five different methods. The first method is the aforementioned non-personalized prediction rule, which employs movie means to yield RMSE=1.053. Henceforth, it will be denoted as MovieAvg. The second method is a correlation-based neighborhood model, which is the most popular approach in the CF literature. As mentioned in Sec. 3, it achieves a RMSE of 0.9406 on the test set, and was named CorNgbr. The third method is the improved neighborhood approach of [2], which we named WgtNgbr and could achieve RMSE= 0.9107 on the test set. Fourth is the SVD latent factor model, with 100 factors thereby achieving RMSE=0.9025 as reported in Table 1. Finally, we consider our most accurate method, the integrated model, with 100 factors, achieving RMSE=0.8870 as shown in Table 2.

Figure 3(top) plots the cumulative distribution of the computed percentile ranks for the five methods over the 384,573 evaluated cases. Clearly, all methods would outperform a random/constant prediction rule, which would have resulted in a straight line connecting the bottom-left and top-right corners. Also, the figure exhibits an ordering of the methods by their strength. In order to achieve a better understanding, let us zoom in on the head of the  $x$ -axis, which represents top-K recommendations. After all, in order to get into the top-K recommendations, a product should be ranked before almost all others. For example, if 600 products are considered, and three of them will be suggested to the user, only those ranked 0.5% or lower are relevant. In a sense, there is no difference between placing a desired 5-star movie at the top 5%, top 20% or top 80%, as none of them is good enough to be presented to the user. Accordingly, Fig. 3(bottom), plots the cumulative ranks distribution between 0% and 2% (top 20 ranked items out of 1000).

As the figure shows, there are very significant differences among the methods. For example, the integrated method has a probability of 0.067 to place a 5-star movie before all other 1000 movies (rank=0%). This is more than three times better than the chance of the MovieAvg method to achieve the same. In addition, it is 2.8 times better than the chance of the popular CorNgbr to achieve the same. The other two methods, WgtNbr and SVD, have a probability of around 0.043 to achieve the same. The practical interpretation is that if about 0.1% of the items are selected to be suggested to

the user, the integrated method has a significantly higher chance to pick a specified 5-star rated movie. Similarly, the integrated method has a probability of 0.157 to place the 5-star movie before at least 99.8% of the random movies (rank $\leq$ 0.2%). For comparison, MovieAvg and CorNgbr have much slimmer chances of achieving the same: 0.050 and 0.065, respectively. The remaining two methods, WgtNgbr and SVD, lie between with probabilities of 0.107 and 0.115, respectively. Thus, if one movie out of 500 is to be suggested, its probability of being a specific 5-stars rated one becomes noticeably higher with the integrated model.

We are encouraged, even somewhat surprised, by the results. It is evident that small improvements in RMSE translate into significant improvements in quality of the top K products. In fact, based on RMSE differences, we did not expect the integrated model to deliver such an emphasized improvement in the test. Similarly, we did not expect the very weak performance of the popular correlation based neighborhood scheme, which could not improve much upon a non-personalized scheme.



**Figure 3: Comparing the performance of five methods on a top-K recommendation task, where a few products need to be suggested to a user. Values on the  $x$ -axis stand for the percentile ranking of a 5-star rated movie; lower values represent more successful recommendations. We experiment with 384,573 cases and show the cumulative distribution of the results. The lower plot concentrates on the more relevant region, pertaining to low  $x$ -axis values. The plot shows that the integrated method has the highest probability of obtaining low values on the  $x$ -axis. On the other hand, the non-personalized MovieAvg method and the popular correlation-based neighborhood method (CorNgbr) achieve the lowest probabilities.**

<sup>3</sup>In this study, the validation set is excluded from the training data.

## 7. DISCUSSION

This work proposed improvements to two of the most popular approaches to Collaborative Filtering. First, we suggested a new neighborhood based model, which unlike previous neighborhood methods, is based on formally optimizing a global cost function. This leads to improved prediction accuracy, while maintaining merits of the neighborhood approach such as explainability of predictions and ability to handle new users without re-training the model. Second, we introduced extensions to SVD-based latent factor models that allow improved accuracy by integrating implicit feedback into the model. One of the models also provides advantages that are usually regarded as belonging to neighborhood models, namely, an ability to explain recommendations and to handle new users seamlessly. In addition, the new neighborhood model enables us to derive, for the first time, an integrated model that combines the neighborhood and the latent factor models. This is helpful for improving system performance, as the neighborhood and latent factor models address the data at different levels and complement each other.

Quality of a recommender system is expressed through multiple dimensions including: accuracy, diversity, ability to surprise with unexpected recommendations, explainability, appropriate top-K recommendations, and computational efficiency. Some of those criteria are relatively easy to measure, such as accuracy and efficiency that were addressed in this work. Some other aspects are more elusive and harder to quantify. We suggested a novel approach for measuring the success of a top-K recommender, which is central to most systems where a few products should be suggested to each user. It is notable that evaluating top-K recommenders significantly sharpens the differences between the methods, beyond what a traditional accuracy measure could show.

A major insight beyond this work is that improved recommendation quality depends on successfully addressing different aspects of the data. A prime example is using implicit user feedback to extend models' quality, which our methods facilitate. Evaluation of this was based on a very limited form of implicit feedback, which was available within the Netflix dataset. This was enough to show a marked improvement, but further experimentation is needed with better sources of implicit feedback, such as purchase/rental history. Other aspects of the data that may be integrated to improve prediction quality are content information like attributes of users or products, or dates associated with the ratings, which may help to explain shifts in user preferences.

### Acknowledgements

The author thanks Suhrid Balakrishnan, Robert Bell and Stephen North for their most helpful remarks.

## 8. REFERENCES

- [1] G. Adomavicius and A. Tuzhilin, "Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions", *IEEE Transactions on Knowledge and Data Engineering* **17** (2005), 634–749.
- [2] R. Bell and Y. Koren, "Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights", *IEEE International Conference on Data Mining (ICDM'07)*, pp. 43–52, 2007.
- [3] R. Bell and Y. Koren, "Lessons from the Netflix Prize Challenge", *SIGKDD Explorations* **9** (2007), 75–79.
- [4] R. M. Bell, Y. Koren and C. Volinsky, "Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems", *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.
- [5] J. Bennet and S. Lanning, "The Netflix Prize", *KDD Cup and Workshop*, 2007. [www.netflixprize.com](http://www.netflixprize.com).
- [6] J. Canny, "Collaborative Filtering with Privacy via Factor Analysis", *Proc. 25th ACM SIGIR Conf.on Research and Development in Information Retrieval (SIGIR'02)*, pp. 238–245, 2002.
- [7] D. Blei, A. Ng, and M. Jordan, "Latent Dirichlet Allocation", *Journal of Machine Learning Research* **3** (2003), 993–1022.
- [8] S. Deerwester, S. Dumais, G. W. Furnas, T. K. Landauer and R. Harshman, "Indexing by Latent Semantic Analysis", *Journal of the Society for Information Science* **41** (1990), 391–407.
- [9] S. Funk, "Netflix Update: Try This At Home", <http://sifter.org/~simon/journal/20061211.html>, 2006.
- [10] D. Goldberg, D. Nichols, B. M. Oki and D. Terry, "Using Collaborative Filtering to Weave an Information Tapestry", *Communications of the ACM* **35** (1992), 61–70.
- [11] J. L. Herlocker, J. A. Konstan and J. Riedl, , "Explaining Collaborative Filtering Recommendations", *Proc. ACM conference on Computer Supported Cooperative Work*, pp. 241–250, 2000.
- [12] J. L. Herlocker, J. A. Konstan, A. Borchers and John Riedl, "An Algorithmic Framework for Performing Collaborative Filtering", *Proc. 22nd ACM SIGIR Conference on Information Retrieval*, pp. 230–237, 1999.
- [13] T. Hofmann, "Latent Semantic Models for Collaborative Filtering", *ACM Transactions on Information Systems* **22** (2004), 89–115.
- [14] D. Kim and B. Yum, "Collaborative Filtering Based on Iterative Principal Component Analysis", *Expert Systems with Applications* **28** (2005), 823–830.
- [15] G. Linden, B. Smith and J. York, "Amazon.com Recommendations: Item-to-item Collaborative Filtering", *IEEE Internet Computing* **7** (2003), 76–80.
- [16] D.W. Oard and J. Kim, "Implicit Feedback for Recommender Systems", *Proc. 5th DELOS Workshop on Filtering and Collaborative Filtering*, pp. 31–36, 1998.
- [17] A. Paterek, "Improving Regularized Singular Value Decomposition for Collaborative Filtering", *Proc. KDD Cup and Workshop*, 2007.
- [18] R. Salakhutdinov, A. Mnih and G. Hinton, "Restricted Boltzmann Machines for Collaborative Filtering", *Proc. 24th Annual International Conference on Machine Learning*, pp. 791–798, 2007.
- [19] R. Salakhutdinov and A. Mnih, "Probabilistic Matrix Factorization", *Advances in Neural Information Processing Systems 20 (NIPS'07)*, pp. 1257–1264, 2008.
- [20] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, "Application of Dimensionality Reduction in Recommender System – A Case Study", *WEBKDD'2000*.
- [21] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, "Item-based Collaborative Filtering Recommendation Algorithms", *Proc. 10th International Conference on the World Wide Web*, pp. 285–295, 2001.
- [22] G. Takacs, I. Pilaszy, B. Nemeth and D. Tikk, "Major Components of the Gravity Recommendation System", *SIGKDD Explorations* **9** (2007), 80–84.
- [23] N. Tintarev and J. Masthoff, "A Survey of Explanations in Recommender Systems", *ICDE'07 Workshop on Recommender Systems and Intelligent User Interfaces*, 2007.

# 5-9: Using the LensKit Evaluator

# Introduction

- LensKit provides a flexible framework for running offline evaluations
- The archetypes provide examples of this
- This video walks through those examples
- In addition to LensKit, examples require:
  - R
  - ggplot2 (`install.packages("ggplot2")` in R)

# Evaluation Scripts

- LensKit evaluator runs eval 'scripts'
- Scripts are written in a Groovy-based DSL
  - Simple case: treat them like config files
  - You don't need to know Groovy well
  - But its full power is available if you want
- Can run from command line (`lenskit-eval` script) or from Maven plugin

# Evaluation Scripts (continued)

- Scripts have various commands
  - crossfold
  - trainTest
  - dumpGraph
- Can also have targets (like Ant)
- Full power of Ant available
  - Groovy AntBuilder (compatible with gant)

# 5-9: Using the LensKit Evaluator



This repository Search

Explore Features Enterprise Blog

Sign up

Sign in

## lenskit / lenskit

Watch 60

Star 286

Fork 131

LensKit recommender toolkit. <http://lenskit.grouplens.org>

4,450 commits

6 branches

43 releases

17 contributors

|   | branch: master ➔ lenskit / +                                      |                          |
|---|---|--------------------------|
| Add tests to check precision-recall functionality |   |                          |
|   | elehack authored 2 days ago                                       | latest commit 3c018791de |
|   | Small code style tweak  | 21 days ago              |
|   | Bump Guava version  | 5 months ago             |
|   | Remove unneeded build system elements                             | 6 months ago             |
|   | Only upload web site on master branch                             | a month ago              |
|   | Update remaining copyright years                                  | 10 months ago            |
|   | Stop using @var tag.  | 2 months ago             |
|   | Add header line support to CLI (#703)                             | 12 days ago              |
|   | Add license header to event test                                  | 12 days ago              |
|   | Better cursor emptiness test                                      | 12 days ago              |
|   | Clean up and fix precision-recall (#685)                          | 2 days ago               |
|   | Rearrange LensKit CLI into org.lenskit.cli package                | 22 days ago              |
|   | Use Graph dev version with aliases                                | 6 months ago             |
|   | Add tests to check precision-recall functionality                 | 2 days ago               |
|   | Improve item-item build logging                                   | 19 days ago              |
|   | Replace several Closers with try-with-resources                   | 26 days ago              |
|   | Stop using SparseVector.fast                                      | 6 months ago             |
|   | Stop using SparseVector.fast                                      | 6 months ago             |
|   | Add Typesafe Config support to lenskit-test                       | 2 months ago             |
|   | use CRLF line endings on gradle.bat                               | 11 months ago            |
|   | Import IntelliJ code style settings into master sources [skip ci] | 22 days ago              |
|   | Disable Java 6 builds for master                                  | 2 months ago             |
|   | contributor list disclaimer [skip ci]                             | 9 months ago             |
|   | Update licensing  | 4 years ago              |
|   | Add Windows build status icon                                     | 5 months ago             |
|   | Add compileOnly configuration to all subprojects                  | 22 days ago              |
|   | Initial Gradle build infrastructure                               | a year ago               |
|   | File cleanup  | 11 months ago            |
|   | Remove unneeded build-support code                                | 2 months ago             |
| README.md   |   |                          |

Code

Issues 117

Pull requests 2

Wiki

Pulse

Graphs

HTTPS clone URL

<https://github.com/lenskit/lenskit> You can clone with [HTTPS](#) or [Subversion](#). 

Clone in Desktop

Download ZIP

## LensKit



LensKit is an implementation of collaborative filtering algorithms and a set of tools for benchmarking them. This readme is about working with the LensKit source code. For more information about LensKit and its documentation, visit the [web site](#) or [wiki](#). You can also find information on the [wiki](#) about how to use LensKit without downloading the source code. If this is your first time working with LensKit we recommend checking out the [Getting Started](#) guide.

LensKit is made available under the GNU Lesser General Public License (LGPL), version 2.1 or later.

## Installation and Dependency Management

LensKit is built and deployed with [Gradle](#) and publishes its artifacts to Maven Central. To install it to the local Maven repository, making it available to other projects using standard Java-based tools, check out this repository and run `./gradlew install`; it is then available to other projects by depending directly on it in your Maven, Gradle, Ivy, or SBT project. The source code can also be checked out and used in most Java IDEs.

Some of the tests require the [Pandas](#) library for Python; if you want to run all the tests (`./gradlew check`), install it with your operating system's package manager (yum, apt-get, brew, etc.), or use `pip`:

```
pip install --user pandas
```

## Working with the Code

To work with the LensKit code, import the Gradle project into your IDE. IntelliJ IDEA includes support for Gradle projects, and this support works well for LensKit in version 13 and later. Gradle plugins are available for other IDEs such as Eclipse.

No particular setup is needed for IntelliJ, which is what most of the LensKit developers use.

## Modules

LensKit is comprised of several modules. The top-level `lenskit` module serves as a container to build them and provide common settings and dependencies. The other modules are as follows:

- `lenskit-api` -- the common, public recommender API exposed by LensKit, independent of its actual implementations.
- `lenskit-test` -- infrastructure and helper code for testing.
- `lenskit-data-structures` -- common data structures used by LensKit. These are split from `-core` so the API can depend on them.
- `lenskit-core` -- the core support code and configuration facilities for the rest of LensKit. It is the entry point for most of what you want to do with LensKit, providing support for configuring and building recommenders.
- `lenskit-knn` -- k-NN recommenders (user-user and item-item collaborative filtering).
- `lenskit-svd` -- the FunkSVD recommender (and eventually real SVD recommenders).
- `lenskit-slopeone` -- Slope-One recommenders.
- `lenskit-eval` -- the evaluation framework and APIs, along with a command line evaluation runner.
- `lenskit-all` -- a metapackage you can depend on to pull in the rest of the LensKit packages.
- `lenskit-cli` -- the LensKit command line interface.
- `lenskit-integration-tests` -- additional integration tests for LensKit.

## Running the Tests

After you make changes, it's good to run the unit tests. You can run many of them from your IDE; run all tests in the `org.grouplens.lenskit` package (and subpackages) across all modules.

To run the full test suite, including data-driven unit tests and integration tests, use Gradle:

```
$ ./gradlew check
```

## Copyright

LensKit is under the following copyright and license:

Copyright 2010-2014 LensKit Contributors.

Work on LensKit has been funded by the National Science Foundation under grants IIS 05-34939, 08-08692, 08-12148, and 10-17697.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.

## Build System Copyright

To ease reuse, the LensKit build system (all `.gradle` files and the contents of the `buildSrc` directory) are licensed under the 3-clause BSD license:

Copyright 2010-2014 LensKit Contributors

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the University of Minnesota nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## Contributing to LensKit

We welcome contribution to LensKit. If you are looking for something to work on, we recommend perusing the open tickets on our [Trac](#) or asking on the mailing list.

We prefer to receive code submissions as GitHub pull requests. To do this:

1. Fork the LensKit repository (`groupLens/lenskit`) on GitHub
2. Push your changes to your fork
3. Submit a pull request via the GitHub web interface

When submitting a pull request via GitHub, you warrant that you either own the code or have appropriate authority to submit it, and license your changes under LensKit's copyright license (LGPLv2.1+ for the main code, 3-clause BSD for the build scripts).



[<< | [Prev](#) | [Index](#) | [Next](#) | >>]**Monday, December 11, 2006***Netflix Update: Try This at Home*[Followup to [this](#)]

Ok, so here's where I tell all about how I (now we) got to be tied for third place on the [netflix prize](#). And I don't mean a sordid tale of computing in the jungle, but rather the actual math and methods. So yes, after reading this post, you too should be able to rank in the top ten or so.

Ur... yesterday's top ten anyway.

My first disclaimer is that our last submission which tied for third place was only actually good enough for ninth place or so. It landed where it did because, just for giggles and grins, we blended results (50/50) with [Jetrays](#) who had a similar score to us at the time.

Second, my friend Vincent has been manning the runs on his desktop machines, diligently fine tuning and squeezing out every last bit of performance possible with whatever controls I could give him (not to mention learning python so he could write scripts to blend submissions and whatnot). In short, almost all my progress since my last post has been due to other people. In the meantime I've implemented a handful of failed attempts at improving the performance, plus one or two minorly successful ones which I'll get to.

Now for the method to the mathness, begining with a review of the problem:

Netflix provided 100M ratings (from 1 to 5) of 17K movies by 500K users. These essentially arrive in the form of a triplet of numbers: (User,Movie,Rating). E.g., one such entry might be (105932,14002,3). Times 100 million. Now go make sense of it. In particular, for (User,Movie,?) not in the database, tell me what the Rating would be--that is, predict how the given User would rate the given Movie.

I'm tempted to get all philosophical on my soap box here and go into ways of thinking about this stuff and modeling vs function mapping approaches, yadda yadda, but I know you all are just here for the math, so I'll save that for the next hapless hosteler who asks me what I do for a living.

For visualizing the problem, it makes sense to think of the data as a big sparsely filled matrix, with users across the top and movies down the side (or vice versa if you feel like transposing everything I say henceforth), and each cell in the matrix either contains an observed rating (1-5) for that movie (row) by that user (column), or is blank meaning you don't know. To quantify "big", sticking with the round numbers, this matrix would have about 8.5 billion entries (number of users times number of movies). Note also that this means you are only given values for one in eighty five of the cells. The rest are all blank.

Netflix has then posed a "quiz" which consists of a bunch of question marks plopped into previously blank slots, and your job is to fill in best-guess ratings in their place. They have chosen mean squared error as the measure of accuracy, which means if you guess 1.5 and the actual rating was 2, you get docked for  $(2-1.5)^2$  points, or 0.25. (In fact they specify root mean squared error, affectionately referred to as rmse, but since they're monotonically related it's all the same and thus it will simply hurt your head less if you ignore the square root at the end.)

One additional tidbit is they also provide a date for both the ratings and the question marks, which aside from giving you more rocks to try to squeeze blood from also implies that any cell in the matrix can potentially have more than one rating in it. This would seem to violate the whole matrix analogy but as it happens that's mere foreplay compared to what follows.

Imagine for a moment that we have the whole shebang--8.5 billion ratings and a lot of weary users. Presumably there are some generalities to be found in there, something more concise and descriptive than 8.5 billion completely independent and unrelated ratings. For instance, any given movie can, to a rough degree of approximation, be described in terms of some basic attributes such as overall quality, whether it's an action movie or a comedy, what stars are in it, and so on. And every user's preferences can likewise be roughly described in terms of whether they tend to rate high or low, whether they prefer action movies or comedies, what stars they like, and so on. And if those basic assumptions are true, then a lot of the 8.5 billion ratings ought to be explainable by a lot less than 8.5 billion numbers, since, for instance, a single number specifying how much action a particular movie has may help explain why a few million action-buffs like that movie.

A fun property of machine learning is that this reasoning works in reverse too: If meaningful generalities can help you represent your

data with fewer numbers, finding a way to represent your data in fewer numbers can often help you find meaningful generalities. Compression is akin to understanding and all that.

In practice this means defining a model of how the data is put together from a smaller number of parameters, and then deriving a method of automatically inferring from the data what those parameters should actually be. In today's foray, that model is called singular value decomposition, which is just a fancy way of saying what I've already eluded to above: We'll assume that a user's rating of a movie is composed of a sum of preferences about the various aspects of that movie.

For example, imagine that we limit it to forty aspects, such that each movie is described only by forty values saying how much that movie exemplifies each aspect, and correspondingly each user is described by forty values saying how much they prefer each aspect. To combine these all together into a rating, we just multiply each user preference by the corresponding movie aspect, and then add those forty leanings up into a final opinion of how much that user likes that movie. E.g., Terminator might be (action=1.2,chickflick=-1,...), and user Joe might be (action=3,chickflick=-1,...), and when you combine the two you get Joe likes Terminator with  $3*1.2 + -1*1 + \dots = 4.6 + \dots$ . Note here that Terminator is tagged as an anti-chickflick, and Joe likewise as someone with an aversion to chickflicks, so Terminator actively scores positive points with Joe for being decidedly un-chickficky. (Point being: negative numbers are ok.) Anyway, all told that model requires 40\*(17K+500K) values, or about 20M -- 400 times less than the original 8.5B.

```
ratingsMatrix[user][movie] = sum (userFeature[f][user] * movieFeature[f][movie]) for f from 1 to 40
```

In matrix terms, the original matrix has been decomposed into two very oblong matrices: the 17,000 x 40 movie aspect matrix, and the 500,000 x 40 user preference matrix. Multiplying those together just performs the products and sums described above, resulting in our approximation to the 17,000 x 500,000 original rating matrix. Singular value decomposition is just a mathematical trick for finding those two smaller matrices which minimize the resulting approximation error--specifically the mean squared error (rather convenient!).

So, in other words, if we take the rank-40 singular value decomposition of the 8.5B matrix, we have the best (least error) approximation we can within the limits of our user-movie-rating model. I.e., the SVD has found our "best" generalizations for us. Pretty neat, eh?

Only problem is, we don't have 8.5B entries, we have 100M entries and 8.4B empty cells. Ok, there's another problem too, which is that computing the SVD of ginormous matrices is... well, no fun. Unless you're into that sort of thing.

But, just because there are five hundred really complicated ways of computing singular value decompositions in the literature doesn't mean there isn't a really simple way too: Just take the derivative of the approximation error and follow it. This has the added bonus that we can choose to simply ignore the unknown error on the 8.4B empty slots.

So, yeah, you mathy guys are rolling your eyes right now as it dawns on you how short the path was.

If you write out the equations for the error between the SVD-like model and the original data--just the given values, not the empties--and then take the derivative with respect to the parameters we're trying to infer, you get a rather simple result which I'll give here in C code to save myself the trouble of formatting the math:

```
userValue[user] += lrate * err * movieValue[movie];
movieValue[movie] += lrate * err * userValue[user];
```

This is kind of like the scene in the Wizard of Oz where Toto pulls back the curtain, isn't it. But wait... let me fluff it up some and make it sound more impressive.

The above code is evaluated for each rating in the training database. Lrate is the learning rate, a rather arbitrary number which I fortuitously set to 0.001 on day one and regretted it every time I tried anything else after that. Err is the residual error from the current prediction. So, the whole routine to train one sample might look like this:

```
/*
 * Where:
 *   real *userValue = userFeature[featureBeingTrained];
 *   real *movieValue = movieFeature[featureBeingTrained];
 *   real lrate = 0.001;
 */
static inline
void train(int user, int movie, real rating)
{
    real err = lrate * (rating - predictRating(movie, user));
    userValue[user] += err * movieValue[movie];
    movieValue[movie] += err * userValue[user];
}
```

Note that predictRating() here would also use userValue and movieValue to do its work, so there's a tight feedback loop in play.

I mention the "static inline" and cram the lrate into err just to make the point that: this is the inside of the inner loop, and every clock cycle counts. My wee laptop is able to do a training pass through the entire data set of 100 million ratings in about seven and a half seconds.

Slightly uglier but more correct, unless you're using an atemporal programming language you will want to do this:

```
uv = userValue[user];
userValue[user] += err * movieValue[movie];
movieValue[movie] += err * uv;
```

Anyway, this will train one feature (aspect), and in particular will find the most prominent feature remaining (the one that will most reduce the error that's left over after previously trained features have done their best). When it's as good as it's going to get, shift it onto the pile of done features, and start a new one. For efficiency's sake, cache the residuals (all 100 million of them) so when you're training feature 72 you don't have to wait for predictRating() to re-compute the contributions of the previous 71 features. You will need 2 Gig of ram, a C compiler, and good programming habits to do this.

There remains the question of what to initialize a new feature to. Unlike backprop and many other gradient descent algorithms, this one isn't really subject to local minima that I'm aware of, which means it doesn't really matter. I initialize both vectors to 0.1, 0.1, 0.1,

0.1, .... Profound, no? (How it's initialized actually does matter a bit later, but not yet...)

The end result, it's worth noting, is exactly an SVD if the training set perfectly covers the matrix. Call it what you will when it doesn't. (If you're wondering where the diagonal scaling matrix is, it gets arbitrarily rolled in to the two side matrices, but could be trivially extracted if needed.)

Before I decide to re-title this entry Much Ado About Nothing, let me pick up the pace now with a host of refinements:

Prior to even starting with the SVD, one can get a good head start by noting the average rating for every movie, as well as the average offset between a user's rating and the movie's average rating, for every user. I.e., the prediction method for this baseline model is:

```
static inline
real predictRating_Baseline(int movie, int user)
{
    return averageRating[movie] + averageOffset[user];
}
```

So, that's the return value of predictRating before the first SVD feature even starts training.

However, even this isn't quite as simple as it appears. You would think the average rating for a movie would just be... its average rating! Alas, Occam's razor was a little rusty that day. Trouble is, to use an extreme example, what if there's a movie which only appears in the training set once, say with a rating of 1. Does it have an average rating of 1? Probably not! In fact you can view that single observation as a draw from a true probability distribution who's average you want... and you can view that true average itself as having been drawn from a probability distribution of averages--the histogram of average movie ratings essentially. If we assume both distributions are Gaussian, then according to my shoddy math the actual best-guess mean should be a linear blend between the observed mean and the apriori mean, with a blending ratio equal to the ratio of variances. That is: If Ra and Va are the mean and variance (squared standard deviation) of all of the movies' average ratings (which defines your prior expectation for a new movie's average rating before you've observed any actual ratings) and Vb is the average variance of individual movie ratings (which tells you how indicative each new observation is of the true mean--e.g., if the average variance is low, then ratings tend to be near the movie's true mean, whereas if the average variance is high, then ratings tend to be more random and less indicative), then:

```
BogusMean = sum(ObservedRatings)/count(ObservedRatings)
K = Vb/Va
BetterMean = [GlobalAverage*K + sum(ObservedRatings)] / [K + count(ObservedRatings)]
```

But in fact K=25 seems to work well so I used that instead. :)

The same principle applies to computing the user offsets. The point here is simply that any time you're averaging a small number of examples, the true average is most likely nearer the apriori average than the sparsely observed average. Note if the number of observed ratings for a particular movie is zero, the BetterMean (best guess) above defaults to the global average movie rating as one would expect.

Moving on:

20 million free parameters is still rather a lot for a training set with only 100 million examples. While it seems like a neat idea to just ignore all those blank spaces in the implicit ratings matrix, the truth is we have some expectations about what's in them, and we can use that to our advantage. As-is, this modified SVD algorithm tends to make a mess of sparsely observed movies or users. To give an example, imagine you have a user who has only rated one movie, say American Beauty. Let's say they give it a 2 while the average is (just making something up) 4.5, and further that their offset is only -1, so we would, prior to even employing the SVD, expect them to rate it 3.5. So the error given to the SVD is -1.5 (the true rating is 1.5 less than we expect). Now imagine that the current movie-side feature, based on broader context, is training up to measure the amount of Action, and let's say that's a paltry 0.01 for American Beauty (meaning it's just slightly more than average). The SVD, recall, is trying to optimize our predictions, which it can do by eventually setting our user's preference for Action to a huge -150.0. I.e., the algorithm naively looks at the one and only example it has of this user's preferences, in the context of the one and only feature it knows about so far (Action), and determines that our user so hates action movies that even the tiniest bit of action in American Beauty makes it suck a lot more than it otherwise might. This is not a problem for users we have lots of observations for because those random apparent correlations average out and the true trends dominate.

So, once again, we need to account for priors. As with the average movie ratings, it would be nice to be able to blend our sparse observations in with some sort of prior, but it's a little less clear how to do that with this incremental algorithm. But if you look at where the incremental algorithm theoretically converges, you get something like:

```
userValue[user] = [sum residual[user,movie]*movieValue[movie]] / [sum (movieValue[movie]^2)]
```

The numerator there will fall in a roughly zero-mean Gaussian distribution when charted over all users, which through various gyrations I won't bore you with leads to:

```
userValue[user] = [sum residual[user,movie]*movieValue[movie]] / [sum (movieValue[movie]^2 + K)]
```

And finally back to:

```
userValue[user] += lrate * (err * movieValue[movie] - K * userValue[user]);
movieValue[movie] += lrate * (err * userValue[user] - K * movieValue[movie]);
```

This is essentially equivalent to penalizing the magnitude of the features, and so is probably related to [Tikhonov regularization](#). The point here is to try to cut down on over fitting, ultimately allowing us to use more features. Last I recall, Vincent liked K=0.02 or so, with well over 100 features (singular vector pairs--if you can still call them that).

Moving on:

As I mentioned a few entries ago, linear models are pretty limiting. Fortunately, we've bastardized the whole matrix analogy so much by now that we aren't really restricted to linear models any more: We can add non-linear outputs such that instead of predicting with:

```
sum (userFeature[f][user] * movieFeature[f][movie]) for f from 1 to 40
```

We can use:

```
sum G(userFeature[f][user] * movieFeature[f][movie]) for f from 1 to 40
```

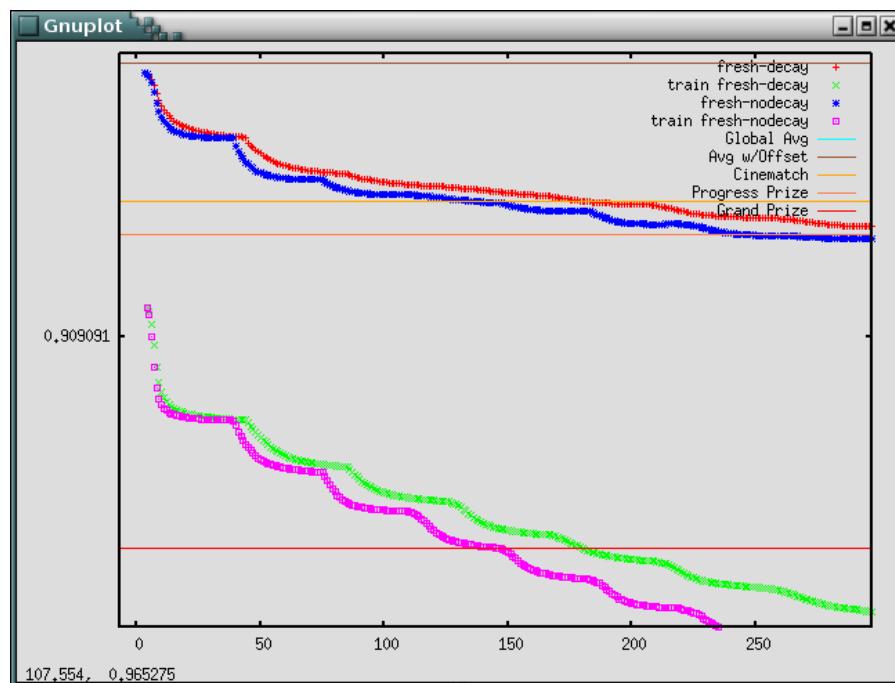
Two choices for G proved useful. One is to simply clip the prediction to the range 1-5 after each component is added in. That is, each feature is limited to only swaying the rating within the valid range, and any excess beyond that is lost rather than carried over. So, if the first feature suggests +10 on a scale of 1-5, and the second feature suggests -1, then instead of getting a 5 for the final clipped score, it gets a 4 because the score was clipped after each stage. The intuitive rationale here is that we tend to reserve the top of our scale for the perfect movie, and the bottom for one with no redeeming qualities whatsoever, and so there's a sort of measuring back from the edges that we do with each aspect independently. More pragmatically, since the target range has a known limit, clipping is guaranteed to improve our performance, and having trained a stage with clipping on we should use it with clipping on. However, I did not really play with this extensively enough to determine there wasn't a better strategy.

A second choice for G is to introduce some functional non-linearity such as a sigmoid. I.e.,  $G(x) = \text{sigmoid}(x)$ . Even if G is fixed, this requires modifying the learning rule slightly to include the slope of G, but that's straightforward. The next question is how to adapt G to the data. I tried a couple of options, including an adaptive sigmoid, but the most general and the one that worked the best was to simply fit a piecewise linear approximation to the true output/output curve. That is, if you plot the true output of a given stage vs the average target output, the linear model assumes this is a nice 45 degree line. But in truth, for the first feature for instance, you end up with a kink around the origin such that the impact of negative values is greater than the impact of positive ones. That is, for two groups of users with opposite preferences, each side tends to penalize more strongly than the other side rewards for the same quality. Or put another way, below-average quality (subjective) hurts more than above-average quality helps. There is also a bit of a sigmoid to the natural data beyond just what is accounted for by the clipping. The linear model can't account for these, so it just finds a middle compromise; but even at this compromise, the inherent non-linearity shows through in an actual-output vs. average-target-output plot, and if G is then simply set to fit this, the model can further adapt with this new performance edge, which leads to potentially more beneficial non-linearity and so on... This introduces new free parameters and again encourages over fitting especially for the later features which tend to represent fairly small groups. We found it beneficial to use this non-linearity only for the first twenty or so features and to disable it after that.

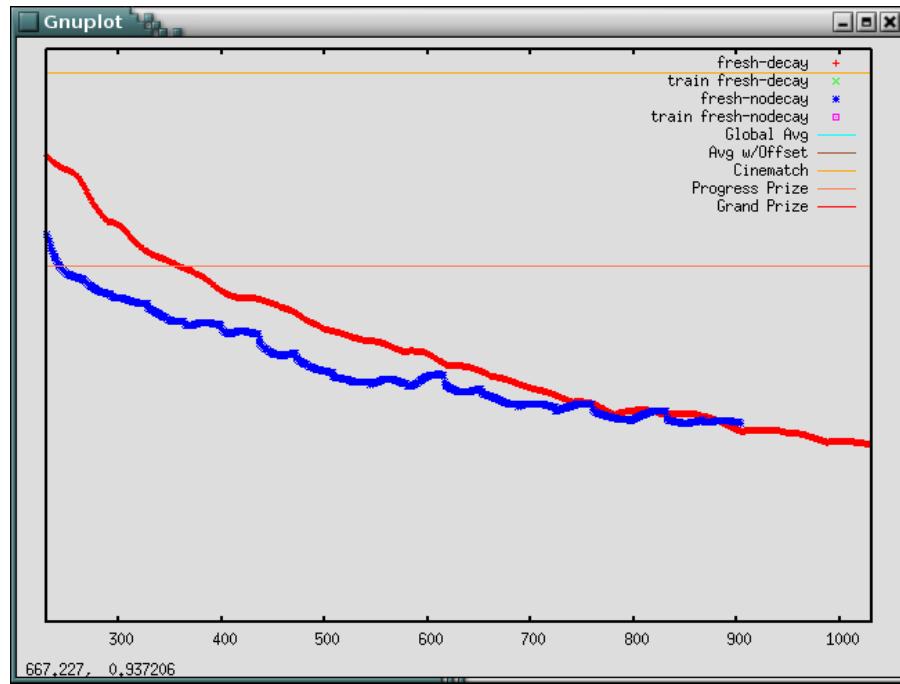
Moving on:

Despite the regularization term in the final incremental law above, over fitting remains a problem. Plotting the progress over time, the probe rmse eventually turns upward and starts getting worse (even though the training error is still inching down). We found that simply choosing a fixed number of training epochs appropriate to the learning rate and regularization constant resulted in the best overall performance. I think for the numbers mentioned above it was about 120 epochs per feature, at which point the feature was considered done and we moved on to the next before it started over fitting. Note that now it does matter how you initialize the vectors: Since we're stopping the path before it gets to the (common) end, where we started will affect where we are at that point. I do wonder if a better regularization method couldn't eliminate overfitting altogether, something like Dirichlet priors in an EM approach--but I tried that and a few others and none worked as well as the above.

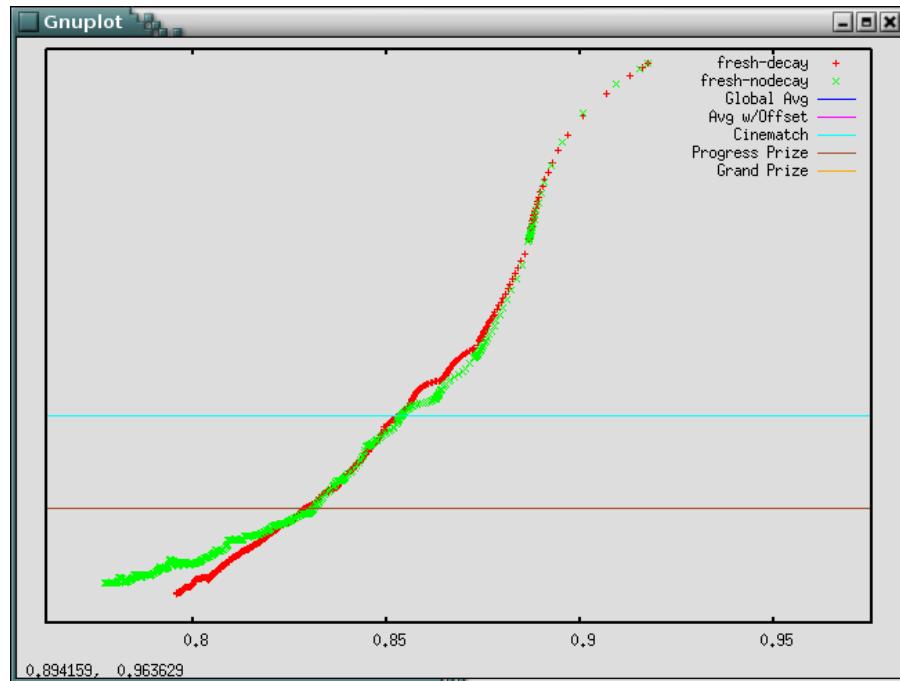
Here is the probe and training rmse for the first few features with and without the regularization term ("decay") enabled.



Same thing, just the probe set rmse, further along where you can see the regularized version pulling ahead:



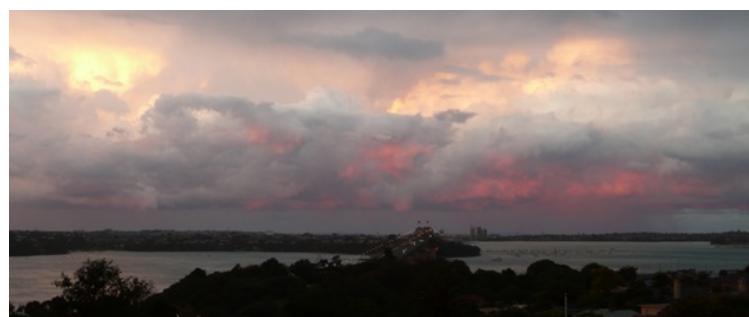
Same plot again, but this time showing probe rmse (vertical) against train rmse (horizontal). Note how the regularized version has better probe performance relative to the training performance:



Anyway, that's about it. I've tried a few other ideas over the last couple of weeks, including a couple of ways of using the date information, and while many of them have worked well up front, none held their advantage long enough to actually improve the final result.

If you notice any obvious errors or have reasonably quick suggestions for better notation or whatnot to make this explanation more clear, let me know. And of course, I'd love to hear what y'all are doing and how well it's working, whether it's improvements to the above or something completely different. Whatever you're willing to share,

-Simon





# Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems

Robert M. Bell, Yehuda Koren and Chris Volinsky

AT&T Labs – Research

180 Park Ave, Florham Park, NJ 07932

{rbell,yehuda,volinsky}@research.att.com

## ABSTRACT

The collaborative filtering approach to recommender systems predicts user preferences for products or services by learning past user-item relationships. In this work, we propose novel algorithms for predicting user ratings of items by integrating complementary models that focus on patterns at different scales. At a local scale, we use a neighborhood-based technique that infers ratings from observed ratings by similar users or of similar items. Unlike previous local approaches, our method is based on a formal model that accounts for interactions within the neighborhood, leading to improved estimation quality. At a higher, regional, scale, we use SVD-like matrix factorization for recovering the major structural patterns in the user-item rating matrix. Unlike previous approaches that require imputations in order to fill in the unknown matrix entries, our new iterative algorithm avoids imputation. Because the models involve estimation of millions, or even billions, of parameters, shrinkage of estimated values to account for sampling variability proves crucial to prevent overfitting. Both the local and the regional approaches, and in particular their combination through a unifying model, compare favorably with other approaches and deliver substantially better results than the commercial Netflix Cinematch recommender system on a large publicly available data set.

## Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications—*Data Mining*

## General Terms

Algorithms

## Keywords

collaborative filtering, recommender systems

## 1. INTRODUCTION

Recommender systems [1] are programs and algorithms that measure the user interest in given items or products to provide personalized recommendations for items that will suit the user's taste. More broadly, recommender systems attempt to profile user preferences and model the interaction between users and products. Increasingly, their excellent ability to characterize and recommend items

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD'07, August 12–15, 2007, San Jose, California, USA.  
Copyright 2007 ACM 978-1-59593-609-7/07/0008 ...\$5.00.

within huge collections represent a computerized alternative to human recommendations.

The growing popularity of e-commerce brings an increasing interest in recommender systems. While users browse a web site, well calculated recommendations expose them to interesting products or services that they may consume. The huge economic potential led some of the biggest e-commerce web, for example web merchant Amazon.com and the online movie rental company Netflix, make the recommender system a salient part of their web sites. High quality personalized recommendations deepen and add another dimension to the user experience. For example, Netflix users can base a significant portion of their movie selection on automatic recommendations tailored to their tastes.

Broadly speaking, recommender systems are based on two different strategies (or combinations thereof). The *content based approach* creates a profile for each user or product to characterize its nature. As an example, a movie profile could include attributes regarding its genre, the participating actors, its box office popularity, etc. User profiles might include demographic information or answers to a suitable questionnaire. The resulting profiles allow programs to associate users with matching products. However, content based strategies require gathering external information that might not be available or easy to collect.

An alternative strategy, our focus in this work, relies only on past user behavior without requiring the creation of explicit profiles. This approach is known as *Collaborative Filtering* (CF), a term coined by the developers of the first recommender system - Tapestry [4]. CF analyzes relationships between users and interdependencies among products, in order to identify new user-item associations. For example, some CF systems identify pairs of items that tend to be rated similarly or like-minded users with similar history of rating or purchasing to deduce unknown relationships between users and items. The only required information is the past behavior of users, which might be, e.g., their previous transactions or the way they rate products. A major appeal of CF is that it is domain free, yet it can address aspects of the data that are often elusive and very difficult to profile using content based techniques. This has led to many papers (e.g., [7]), research projects (e.g., [9]) and commercial systems (e.g., [11]) based on CF.

In a more abstract manner, the CF problem can be cast as missing value estimation: we are given a user-item matrix of scores with many missing values, and our goal is to estimate the missing values based on the given ones. The known user-item scores measure the amount of interest between respective users and items. They can be explicitly given by users that rate their interest in certain items or might be derived from historical purchases. We call these user-item scores *ratings*, and they constitute the input to our algorithm.

## Research Track Paper

In October 2006, the online movie renter, Netflix, announced a CF-related contest – *The Netflix Prize* [10]. Within this contest, Netflix published a comprehensive dataset including more than 100 million movie ratings, which were performed by about 480,000 real customers (with hidden identities) on 17,770 movies. Competitors in the challenge are required to estimate a few million ratings. To win the “grand prize,” they need to deliver a 10% improvement in the prediction root mean squared error (RMSE) compared with the results of Cinematch, Netflix’s proprietary recommender system. This real life dataset, which is orders of magnitudes larger than previous available datasets for in CF research, opens new possibilities and has the potential to reduce the gap between scientific research and the actual demands of commercial CF systems. The algorithms in this paper were tested and motivated by the Netflix data, but they are not limited to that dataset, or to user-movie ratings in general.

Our approach combines several components; each of which models the data at a different scale in order to estimate the unknown ratings. The *global component* utilizes the basic characteristics of users and movies, in order to reach a first-order estimate of the unknown ratings. At this stage the interaction between items and users is minimal. The more refined, *regional component*, defines –for each item and user– *factors* that strive to explain the way in which users interact with items and form ratings. Each such factor corresponds to a viewpoint on the given data along which we measure users and items simultaneously. High ratings correspond to user-item pairs with closely matching factor values. The *local component* models the relationships between similar items or users, and derive unknown ratings from values in user/item neighborhoods.

The main contributions detailed in this paper are the following:

- Derivation of a local, neighborhood-based approach where the interpolation weights solve a formal model. This model simultaneously accounts for relationships among all neighbors, rather than only for two items (or users) at a time. Furthermore, we extend the model to allow item-item weights to vary by user, and user-user weights to vary by item.
- A new iterative algorithm for an SVD-like factorization that avoids the need for imputation, thereby being highly scalable and adaptable to changes in the data. We extend this model to include an efficient integration of neighbors’ information.
- Shrinkage of parameters towards common values to allow utilization of very rich models without overfitting.
- Derivation of confidence scores that facilitate combining scores across algorithms and indicate the degree of confidence in predicted ratings.

Both the local and regional components outperformed the published results by Netflix’s proprietary Cinematch system. Additionally, combination of the components produced even better results, as we explain later.

The paper explains the three components, scaling up from the local, neighborhood-based approach (Section 3), into the regional, factorization-based approach (Section 4). Discussion of some related works is split between Sections 3 and 4, according to its relevancy. Experimental results, along with discussion of confidence scores is given in Section 5. We begin in Section 2 by describing the general data framework, the methodology for parameter shrinkage, and the approach for dealing with global effects.

## 2. GENERAL FRAMEWORK

### 2.1 Notational conventions

We are given ratings about  $m$  users and  $n$  items, arranged in an  $m \times n$  matrix  $R = \{r_{ui}\}_{1 \leq u \leq m, 1 \leq i \leq n}$ . We anticipate three characteristics of the data that may complicate prediction. First,

the numbers of users and items may be very large, as in the Netflix data, with the former likely much larger than the latter. Second, an overwhelming portion of the user-item matrix (e.g., 99%) may be unknown. Sometimes we refer to matrix  $R$  as a sparse matrix, although its sparsity pattern is driven by containing many unknown values rather than the common case of having many zeros. Third, the pattern of observed data may be very nonrandom, i.e., the amount of observed data may vary by several orders of magnitude among users or among items.

We reserve special indexing letters for distinguishing users from items: for users  $u, v$ , and for items  $i, j, k$ . The known entries – those  $(u, i)$  pairs for which  $r_{ui}$  is known – are stored in the set  $\mathcal{K} = \{(u, i) \mid r_{ui} \text{ is known}\}$ .

### 2.2 Shrinking estimated parameters

The algorithms described in Sections 3 and 4 can each lead to estimating massive numbers of parameters (not including the individual ratings). For example, for the Netflix data, each algorithm estimates at least ten million parameters. Because many of the parameter estimates are based on small sample sizes, it is critical to take steps to avoid overfitting the observed data.

Although cross validation is a great tool for tuning a few parameters, it is inadequate for dealing with massive numbers of parameters. Cross validation works very well for determining whether the use of a given parameter improves predictions relative to other factors. However, that all-or-nothing choice does not scale up to selecting or fitting many parameters at once. Sometimes the amount of data available to fit different parameters will vary widely, perhaps by orders of magnitude. The idea behind “shrinkage” is to impose a penalty for those parameters that have less data associated with them. The penalty is to shrink the parameter estimate towards a null value, often zero. Shrinkage is a more continuous alternative to parameter selection, and as implemented in statistical methods like ridge and lasso regression [16] has been shown in many cases to improve predictive accuracy.

We can approach shrinkage from a Bayesian perspective. In the typical Bayesian paradigm, the best estimator of a parameter is the posterior mean, a linear combination of the prior mean of the parameter (null value) and an empirical estimator based fully on the data. The weights on the linear combination depend on the ratio of the prior variance to the variance of the empirical estimate. If the estimate is based on a small amount of data, the variance of that estimate is relatively large, and the data-based estimates are shrunk toward the null value. In this way shrinkage can simultaneously “correct” all estimates based on the amount of data that went into calculating them.

For example, we can calculate similarities between two items based on the correlation for users who have rated them. Because we expect pairwise correlations to be centered around zero, the empirical correlations are shrunk towards zero. If the empirical correlation of items  $i$  and  $j$  is  $s_{ij}$ , based on  $n_{ij}$  observations, then the shrunk correlation has the form  $n_{ij}s_{ij}/(n_{ij} + \beta)$ . For fixed  $\beta$ , the amount of shrinkage is inversely related to the number of users that have rated both items: the higher  $n_{ij}$  is, the more we “believe” the estimate, and the less we shrink towards zero. The amount of shrinkage is also influenced by  $\beta$ , which we generally tune using cross validation.

### 2.3 Removal of global effects

Before delving into more involved methods, a meaningful portion of the variation in ratings can be explained by using readily available variables that we refer to as global effects. The most obvious global effects correspond to item and user effects – i.e., the

tendency for ratings of some items or by some users to differ systematically from the average. These effects are removed from the data, and the subsequent algorithms work with the remaining residuals. The process is similar to double-centering the data, but each step must shrink the estimated parameters properly.

Other global effects that can be removed are dependencies between the rating data and some known attributes. While a pure CF framework does not utilize content associated with the data, when such content is given, there is no reason not to exploit it. An example of a universal external attribute is the dates of the ratings. Over time, items go out of fashion, people may change their tastes, their rating scales, or even their “identities” (e.g., a user may change from being primarily a parent to primarily a child from the same family). Therefore, it can be beneficial to regress against time the ratings by each user, and similarly, the ratings for each item. This way, time effects can explain additional variability that we recommend removing before turning to more involved methods.

### 3. NEIGHBORHOOD-BASED ESTIMATION

#### 3.1 Previous work

The most common approach to CF is the local, or neighborhood-based approach. Its original form, which was shared by virtually all earlier CF systems, is the user-oriented approach; see [7] for a good analysis. Such user-oriented methods estimate unknown ratings based on recorded ratings of like minded users. More formally, in order to estimate the unknown rating  $r_{ui}$ , we resort to a set of users  $N(u; i)$ , which tend to rate similarly to  $u$  (“neighbors”), and actually rated item  $i$  (i.e.,  $r_{vi}$  is known for each  $v \in N(u; i)$ ). Then, the estimated value of  $r_{ui}$  is taken as a weighted average of the neighbors’ ratings:

$$\frac{\sum_{v \in N(u; i)} s_{uv} r_{vi}}{\sum_{v \in N(u; i)} s_{uv}} \quad (1)$$

The similarities – denoted by  $s_{uv}$  – play a central role here as they are used both for selecting the members of  $N(u; i)$  and for weighting the above average. Common choices are the Pearson correlation coefficient and the closely related cosine similarity. Prediction quality can be improved further by correcting for user-specific means, which is related to the global effects removal discussed in Subsection 2.3. More advanced techniques account not only for mean translation, but also for scaling, leading to modest improvements; see, e.g., [15].

An analogous alternative to the user-oriented approach is the item-oriented approach [15, 11]. In those methods, a rating is estimated using known ratings made by the same user on similar items. Now, to estimate the unknown  $r_{ui}$ , we identify a set of neighboring items  $N(i; u)$ , which other users tend to rate similarly to their rating of  $i$ . Analogous to above, all items in  $N(i; u)$  must have been rated by  $u$ . Then, in parallel to (1), the estimated value of  $r_{ui}$  is taken as a weighted average of the ratings of neighboring items:

$$\frac{\sum_{j \in N(i; u)} s_{ij} r_{uj}}{\sum_{j \in N(i; u)} s_{ij}} \quad (2)$$

As with the user-user similarities, the item-item similarities (denoted by  $s_{ij}$ ) are typically taken as either correlation coefficients or cosine similarities. Sarwar et al. [15] recommend using an adjusted cosine similarity on ratings that had been translated by deducting user-means. They found that item-oriented approaches deliver better quality estimates than user-oriented approaches while allowing more efficient computations. This is because there are typically

significantly fewer items than users, which allows pre-computing all item-item similarities for retrieval as needed.

Neighborhood-based methods became very popular, because they are intuitive and relatively simple to implement. In our eyes, a main benefit is their ability to provide a concise and intuitive justification for the computed predictions, presenting the user a list of similar items that she has previously rated, as the basis for the estimated rating. This allows her to better assess its relevance (e.g., downgrade the estimated rating if it is based on an item that she no longer likes) and may encourage the user to alter outdated ratings.

However, neighborhood-based methods share some significant disadvantages. The most salient one is the heuristic nature of the similarity functions ( $s_{uv}$  or  $s_{ij}$ ). Different rating algorithms use somewhat different similarity measures; all are trying to quantify the elusive notion of the level of user- or item-similarity. We could not find any fundamental justification for the chosen similarities.

Another problem is that previous neighborhood-based methods do not account for interactions among neighbors. Each similarity between an item  $i$  and a neighbor  $j \in N(i; u)$ , and consequently its weight in (2), is computed independently of the content of  $N(i; u)$  and the other similarities:  $s_{ik}$  for  $k \in N(i; u) - \{j\}$ . For example, suppose that our items are movies, and the neighbors set contains three movies that are very close in their nature (e.g., sequels such as “Lord of the Rings 1–3”). An algorithm that ignores the similarity of the three movies when predicting the rating for another movie, may triple count the available information and disproportionately magnify the similarity aspect shared by these three very similar movies. A better algorithm would account for the fact that part of the prediction power of a movie such as “Lord of the Rings 3” may already be captured by “Lord of the Rings 1–2”, and vice versa, and thus discount the similarity values accordingly.

#### 3.2 Modeling neighborhood relations

We overcome the above problems of neighborhood-based approaches by relying on a suitable model. To our best knowledge, this is the first time that a neighborhood-based approach is derived as a solution to a model, and allows simultaneous, rather than isolated, computation of the similarities.

In the following discussion we derive an item-oriented approach, but a parallel idea was successfully applied in a user-oriented fashion. Also, we use the term “weights” ( $w_{ij}$ ) rather than “similarities”, to denote the coefficients in the weighted average of neighborhood ratings. This reflects the fact that for a fixed item  $i$ , we compute all related  $w_{ij}$ ’s simultaneously, so that each  $w_{ij}$  may be influenced by other neighbors. In what follows, our target is always to estimate the unknown rating by user  $u$  of item  $i$ , that is  $r_{ui}$ .

The first phase of our method is neighbor selection. Among all items rated by  $u$ , we select the  $g$  most similar to  $i - N(i; u)$ , by using a similarity function such as the correlation coefficient, properly shrunk as described in Section 2.2. The choice of  $g$  reflects a tradeoff between accuracy and efficiency; typical values lie in the range of 20–50; see Section 5.

After identifying the set of neighbors, we define the cost function (the “model”), whose minimization determines the weights. We look for the set of interpolation weights  $\{w_{ij} | j \in N(i; u)\}$  that will enable the best prediction rule of the form

$$r_{ui} = \frac{\sum_{j \in N(i; u)} w_{ij} r_{uj}}{\sum_{j \in N(i; u)} w_{ij}} \quad (3)$$

We restrict all interpolation weights to be nonnegative, that is,  $w_{ij} \geq 0$ , which allows simpler formulation and, importantly, has proved beneficial in preventing overfitting.

## Research Track Paper

We denote by  $U(i)$  the set of users who rated item  $i$ . Certainly, our target user,  $u$ , is not within this set. For each user  $v \in U(i)$ , denote by  $N(i; u, v)$ , the subset of  $N(i; u)$  that includes the items rated by  $v$ . In other words,  $N(i; u, v) = \{j \in N(i; u) | (v, j) \in \mathcal{K}\}$ . For each user  $v \in U(i)$ , we seek weights that will perfectly interpolate the rating of  $i$  from the ratings of the given neighbors:

$$r_{vi} = \frac{\sum_{j \in N(i; u, v)} w_{ij} r_{vj}}{\sum_{j \in N(i; u, v)} w_{ij}} \quad (4)$$

Notice that the only unknowns here are the weights ( $w_{ij}$ ). We can find many perfect interpolation weights for each particular user, which will reconstruct  $r_{vi}$  from the  $r_{vj}$ 's. After all, we have one equation and  $|N(i; u, v)|$  unknowns. The more interesting problem is to find weights that simultaneously work well for all users. This leads to a least squares problem:

$$\min_w \sum_{v \in U(i)} \left( r_{vi} - \frac{\sum_{j \in N(i; u, v)} w_{ij} r_{vj}}{\sum_{j \in N(i; u, v)} w_{ij}} \right)^2 \quad (5)$$

Expression (5) is unappealing because it treats all squared deviations (or users) equally. First, we want to give more weight to users that rated many items of  $N(i; u)$ . Second, we want to give more weight to users who rated items most similar to  $i$ . We account for these two considerations by weighting the term associated with user  $v$  by  $\left(\sum_{j \in N(i; u, v)} w_{ij}\right)^\alpha$ , which signifies the relative importance of user  $v$ . To simplify subsequent derivation, we chose  $\alpha = 2$ , so the sum to be minimized is:

$$\min_w \sum_{v \in U(i)} c_v \left( r_{vi} - \frac{\sum_{j \in N(i; u, v)} w_{ij} r_{vj}}{\sum_{j \in N(i; u, v)} w_{ij}} \right)^2 / \sum_{v \in U(i)} c_v \quad (6)$$

where  $c_v = \left(\sum_{j \in N(i; u, v)} w_{ij}\right)^2$ .

At this point, we switch to matrix notation. For notational convenience assume that the  $g$  items from which we interpolate the rating of  $i$  are indexed by  $1, \dots, g$ , and arranged within  $w \in \mathbb{R}^g$ . Let us define two  $g \times g$  matrices  $A$  and  $B$ , where the  $(j, k)$ -entry sums over all users in  $U(i)$  that rated both  $j$  and  $k$ , as follows:

$$A_{jk} = \sum_{v \in U(i) \text{ s.t. } j, k \in N(i; u, v)} (r_{vj} - r_{vi})(r_{vk} - r_{vi}) \quad (7)$$

$$B_{jk} = \sum_{v \in U(i) \text{ s.t. } j, k \in N(i; u, v)} 1 \quad (8)$$

Using these matrices, we can recast problem (6) in the equivalent form:

$$\min_w \frac{w^T Aw}{w^T Bw} \quad (9)$$

Notice that both  $A$  and  $B$  are symmetric positive semidefinite matrices as they correspond to squared sums (numerator and denominator of (6)). It can be shown analytically that the optimal solution of the problem is given by solving the generalized eigenvector equation  $Aw = \lambda Bw$  and taking the weights as the eigenvector associated with the smallest eigenvalue. However, we requested that the interpolation weights are nonnegative, so we need to add a non-negativity constraint  $w \geq 0$ . In addition, since  $\frac{w^T Aw}{w^T Bw}$  is invariant under scaling of  $w$ , we fix the scale of  $w$  obtaining the equivalent problem:

$$\min_w w^T Aw \text{ s.t. } w^T Bw = 1, w \geq 0 \quad (10)$$

It is no longer possible to find an analytic solution to the problem in the form of an eigenvector. To solve this problem, one

can resort to solving a series of quadratic programs, by linearizing the quadratic constraint  $w^T Bw = 1$ , using the substitution  $x \leftarrow Bw$  and the constraint  $x^T w = 1$ . This typically leads to convergence with 3–4 iterations. In practice, we prefer a slightly modified method that does not require such an iterative process.

### 3.2.1 A revised model

A major challenge that every CF method faces is the sparseness and the non-uniformity of the rating data. Methods must account for the fact that almost all user-item ratings are unknown, and the known ratings are unevenly distributed so that some users/items are associated with many more ratings than others. The method described above avoids these issues, by relying directly on the known ratings and by weighting the importance of each user according to his support in the data. We now describe an alternative approach that initially treats the data as if it is dense, but accounts for the sparseness by averaging and shrinking.

If all ratings of users that rated  $i$  were known, the problem of interpolating the rating of  $i$  from ratings of other items – as in Eq. (3) – is related to multivariate regression, where we are looking for weights that best interpolate the vector associated with our item  $i$ , from the neighboring items  $N(i; u)$ . In this case, problem (5) would form an adequate model, and the user-weighting that led to the subsequent refined formulations of the problem would no longer be necessary. To avoid overfitting, we still restrict the weights to be positive and also fix their scale. Using our previous matrix notation, this leads to the quadratic program:

$$\min_w w^T Aw \text{ s.t. } \sum_i w_i = 1, w \geq 0 \quad (11)$$

Recall that the matrix  $A$  was defined in (7). In the hypothetical dense case, when all ratings by person  $v$  are known, the condition  $j, k \in N(i; u, v)$  is always true, and each  $A_{jk}$  entry is based on the full users set  $U(i)$ . However, in the real, sparse case, each  $A_{jk}$  entry is based on a different set of users that rated both  $j$  and  $k$ . As a consequence, different  $A_{jk}$  entries might have very different scales depending on the size of their support. We account for this, by replacing the sum that constitutes  $A_{jk}$ , with a respective average whose magnitude is not sensitive to the support. In particular, since the support of  $A_{jk}$  is exactly  $B_{jk}$  as defined in (8), we replace the matrix  $A$ , with the matching  $g \times g$  matrix  $A'$ , defined as:

$$A'_{jk} = \frac{A_{jk}}{B_{jk}}$$

This is still not enough for overcoming the sparseness issue. Some  $A'_{jk}$  entries might rely on a very low support (low corresponding  $B_{jk}$ ), so their values are less reliable and should be shrunk towards the average across  $(j, k)$ -pairs. Thus, we compute the average entry value of  $A'$ , which is denoted as  $avg = \sum_{j,k} A'_{jk}/(g^2)$ , and define the corresponding  $g \times g$  matrix  $\hat{A}$ :

$$\hat{A}_{jk} = \frac{B_{jk} \cdot A'_{jk} + \beta \cdot avg}{B_{jk} + \beta}$$

The non-negative parameter  $\beta$  controls the extent of the shrinkage. We typically use values between 10 and 100. A further improvement is achieved by separating the diagonal and non-diagonal entries, accounting for the fact that the diagonal entries are expected to have an inherently higher average because they sum only non-negative values.

The matrix  $\hat{A}$  approximates the matrix  $A$  and thus replaces it in problem (11). One could use a standard quadratic programming solver to derive the weights. However, it is beneficial to exploit

```

NonNegativeQuadraticOpt ( $A \in \mathbb{R}^{k \times k}$ ,  $b \in \mathbb{R}^k$ )
% Minimize  $x^T Ax - 2b^T x$  s.t.  $x \geq 0$ 

do
     $r \leftarrow Ax - b$  % the residual, or “steepest gradient”
    % find active variables - those that are pinned because of
    % nonnegativity constraint, and set respective  $r_i$ ’s to zero
    for  $i = 1, \dots, k$  do
        if  $x_i = 0$  and  $r_i < 0$  then
             $r_i \leftarrow 0$ 
        end if
    end for
     $\alpha \leftarrow \frac{r^T r}{r^T A r}$  % max step size
    % adjust step size to prevent negative values:
    for  $i = 1, \dots, k$  do
        if  $r_i < 0$  then
             $\alpha \leftarrow \min(\alpha, -x_i/r_i)$ 
        end if
    end for
     $x \leftarrow x + \alpha r$ 
    while  $\|r\| < \epsilon$  % stop when residual is close to 0
    return  $x$ 

```

**Figure 1: Minimizing a quadratic function with non-negativity constraints**

the simple structure of the constraints. First, we inject the equality constraint into the cost function, so we actually optimize:

$$\min_w w^T \hat{A}w + \lambda \left( \sum_i w_i - 1 \right)^2 \text{ s.t. } w \geq 0 \quad (12)$$

We construct a  $g \times g$  matrix  $\hat{C}_{jk} = \hat{A}_{jk} + \lambda$  and a vector  $\hat{b} \in \mathbb{R}^g = (\lambda, \lambda, \dots, \lambda)^T$ , so that our problem becomes  $\min_w w^T \hat{C}w - 2\hat{b}^T w$  s.t.  $w \geq 0$ . Higher values of the parameter  $\lambda$ , impose stricter compliance to the  $\sum_i w_i = 1$  constraint and tend to increase running time. We used  $\lambda = 1$ , where estimation quality was uniform across a wide range of  $\lambda$  values, and strict compliance to the  $\sum_i w_i = 1$  constraint was not beneficial. Now, all constraints have a one sided fixed boundary (namely, 0), reaching a simplified quadratic program that we solve by calling the function NonNegativeQuadraticOpt( $\hat{C}, \hat{b}$ ), which is given in Figure 1. The function is based on the principles of the Gradient Projection method; see, e.g., [12]. The running time of this function depends on  $g$ , which is a small constant independent of the magnitude of the data, so it is not the computational bottleneck in the process.

There is a performance price to pay for tailoring the interpolation weights to the specific neighbors set from which we interpolate the query rating. The main computational effort involves scanning the relevant ratings while building the matrices  $A$  and  $B$ . This makes our neighborhood-based method slower than previous methods, where all weights (or similarities) could have been pre-computed. A related approach that vastly improves running time is described in a newer work [2].

### 3.3 Integrating user-user similarities

The fact that we tailor the computation of the interpolation weights to the given  $r_{ui}$  query opens another opportunity. We can weight all users by their similarity to  $u$ . That is, we insert the user-user similarities ( $s_{uv}$ ) into (5), yielding the expression:

$$\min_w \sum_{v \in U(i)} s_{uv} \left( r_{vi} - \frac{\sum_{j \in N(i; u, v)} w_{ij} r_{vj}}{\sum_{j \in N(i; u, v)} w_{ij}} \right)^2 \quad (13)$$

And consequently, we redefine the  $A$  and  $B$  matrices as:

$$A_{jk} = \sum_{v \in U(i) \text{ s.t. } j, k \in N(i; u, v)} s_{uv} (r_{vj} - r_{vi})(r_{vk} - r_{vi})$$

$$B_{jk} = \sum_{v \in U(i) \text{ s.t. } j, k \in N(i; u, v)} s_{uv}$$

Recall that  $s_{uv}$  is a similarity measure between users  $u$  and  $v$ , properly shrunk as described in Section 2.2. Usually, this would be a squared correlation coefficient, or the inverse of the Euclidean distance. Incorporating these similarities into our equations means that the inferred relationship between items depends not only on the identity of the items, but also on the type of user. For example, some users put a lot of emphasis on the actors participating in a movie, while for other users the plot of the movie is much more important than the identity of actors. Certainly, interpolation weights must vary a lot between such different kinds of users. In practice, we found this enhancement of the method significant in improving the prediction accuracy.

To summarize, we build on the intuitive appeal of neighborhood-based approaches, which can easily explain their ratings to the user. Our method differs from previous approaches by being the solution of an optimization problem. Although this leads to more extensive computational effort, it introduces two important aspects that were not addressed in previous works: First, all interpolation weights used for a single prediction are interdependent, allowing consideration of interactions involving many items, not only pairs. Second, item-item interpolation weights depend also on the given user. That is, the relationships between items are not static, but depend on the characteristics of the user. Another benefit of relying on a cost function is the natural emergence of confidence scores for estimating prediction accuracy, as we discuss later in Subsection 5.4.

Our description assumed an item-oriented approach. However, we have used the same approach to produce user-user interpolation weights. This simply requires switching the roles of users and items throughout the algorithm. Despite the fact that the method is computationally intensive, we could apply it successfully to the full Netflix Data on a desktop PC. Our results, which exceeded those of Netflix’s Cinematch, are reported in Section 5.

## 4. FACTORIZATION-BASED ESTIMATION

### 4.1 Background

Now we move up from the local, neighborhood-based approach, to a “regional” approach where we compute a limited set of features that characterize all users and items. These features allow us to link users with items and estimate the associated ratings. For example, consider user-movie ratings. In this case, regional features might be movie genres. One of the features could measure the fitting into the action genre, while another feature could measure fitting into the comedy genre. Our goal would be to place each movie and each user within these genre-oriented scales. Then, when given a certain user-movie pair, we estimate the rating by the closeness of the features representing the movie and the user.

Ranking users and items within prescribed features, such as movie genres, pertains to content-based methods, which requires additional external information on items and users beyond the past ratings. This might be a very complicated data gathering and cleaning task. However, our goal is to uncover latent features of the given data that explain the ratings, as a surrogate for the external information. This can be achieved by employing matrix factorization techniques such as Singular Value Decomposition (SVD) or Principal Component Analysis (PCA); in the context of information retrieval this is widely known as Latent Semantic Indexing [3].

Given an  $m \times n$  matrix  $R$ , SVD computes the best rank- $f$  approximation  $R^f$ , which is defined as the product of two rank- $f$  matrices  $P_{m \times f}$  and  $Q_{n \times f}$ , where  $f \leq m, n$ . That is,  $R^f = PQ^T$  minimizes the Frobenius norm  $\|R - R^f\|_F$  among all rank- $f$  matrices. In this sense, the matrix  $R^f$  captures the  $f$  most prominent features of the data, leaving out less significant portion of the data that might be mere noise.

It is important to understand that the CF context requires a unique application of SVD, due to the fact that most entries of  $R$  are unknown. Usually SVD, or PCA, are used for reducing the dimensionality of the data and lessen its representation complexity by providing a succinct description thereof. Interestingly, for CF we are utilizing SVD for an almost opposite purpose – to extend the given data by filling in the values of the unknown ratings. Each unknown rating,  $r_{ui}$  is estimated as  $R_{ui}^f$ , which is a dot product of the  $u$ -th row of  $P$  with the  $i$ -th row of  $Q$ . Consequently, we refer to  $P$  as the *user factors* and to  $Q$  as the *item factors*.

Beyond the conceptual difference in the way we use SVD, we also face a unique computational difficulty due to the sparsity issue. SVD computation can work only when all entries of  $R$  are known. In fact, the goal of SVD is not properly defined when some entries of  $R$  are missing.

Previous works on adopting SVD- or PCA-based techniques for CF coped with these computational difficulties. Eigenstate [5] uses PCA factorization combined with recursive clustering in order to estimate ratings. Notably, they overcome the sparsity issue by profiling the taste of each user using a universal query that all users must answer. This way, they define a gauge set containing selected representative items for which the ratings of all users must be known. Restricting the rating matrix to the gauge set results in a dense matrix, so conventional PCA techniques can be used for its factorization. However, in practice, obtaining such a gauge set rated by each user, may not be feasible. Moreover, even when obtaining a gauge set is possible, it neglects almost all ratings (all those outside the gauge set), making it likely to overlook much of the given information, thereby degrading prediction accuracy.

An alternative approach is to rely on imputation to fill in missing ratings and make the rating matrix dense. For example, Sarwar et al. [14] filled the missing ratings with a normalized average rating of the related item. Later, Kim and Yum [8] suggested a more involved iterative method, where SVD is iteratively performed while improving the accuracy of imputed values based on results of prior iterations. In a typical CF application, where the imputed ratings significantly outnumber the original ratings, those methods relying on imputation risk distorting the data due to inaccurate imputation. Furthermore, from the computational viewpoint, imputation can be very expensive requiring one to explicitly deal with each user-item combination, therefore significantly increasing the size of the data. This might be impractical for comprehensive datasets (such as the Netflix data), where the number of users may exceed a million with more than ten thousands items.

## 4.2 An EM approach to PCA

We propose a method that avoids the need for a gauge set or for imputation, by working directly on the sparse set of known ratings. Since SVD is not well defined on such sparse matrices, we resort to SVD-generalizations that can handle unknown values. In particular we were inspired by Roweis [13], who described an EM algorithm for PCA, to which we now turn.

The common way to compute the PCA of a matrix  $R$  is by working on its associated covariance matrix. However, Roweis suggested a completely different approach, which eventually leads to the same PCA factorization (up to orthogonalization). Recall that

we try to compute rank- $f$  matrices  $Q$  and  $P$  that will minimize  $\|R - PQ^T\|_F$ . Now, we can fix the matrix  $P$  as some matrix  $\hat{P}$ , such that minimization of  $\|R - PQ^T\|_F$  would be equivalent to the least squares solution of  $R = \hat{P}Q^T$ . Analogously, we can fix  $Q$  as  $\hat{Q}$ , so our minimization problem becomes the least squares solution of  $R = U\hat{Q}^T$ . These least squares problems can be minimized by setting  $Q^T = (\hat{P}^T \hat{P})^{-1} \hat{P}^T R$  and  $P = R\hat{Q}(\hat{Q}^T \hat{Q})^{-1}$ , leading to an iterative process that alternately recomputes the matrices  $P$  and  $Q$ , as follows:

$$Q^T \leftarrow (\hat{P}^T \hat{P})^{-1} \hat{P}^T R \quad (14)$$

$$P \leftarrow RQ(\hat{Q}^T \hat{Q})^{-1} \quad (15)$$

It can be shown that the only possible minimum is the global one, so that  $P$  and  $Q$  must converge to the true SVD subspace [13].

One of the advantages of this iterative SVD computation is its ability to deal with missing values. Roweis proposed to treat the missing values in  $R$  as unknowns when obtaining the least squares solution of  $R = \hat{P}Q^T$ , which is still solvable by standard techniques. This approach actually uses imputation, by filling in the missing values of  $R$  as part of the iterative process. This would be infeasible for large datasets where the number of all possible user-item ratings is huge. Therefore, we modify Roweis' idea to enable it to deal with many missing values, while avoiding imputation.

## 4.3 Our approach

We would like to minimize the squared error between the factors-based estimates and known ratings, which is:

$$\text{Err}(P, Q) \stackrel{\text{def}}{=} \sum_{(u,i) \in \mathcal{K}} (r_{ui} - p_u^T q_i)^2 \quad (16)$$

Here,  $p_u$  is the  $u$ -th row of  $P$ , which corresponds to user  $u$ . Likewise,  $q_i$  is the  $i$ -th row of  $Q$ , which corresponds to item  $i$ . Similar to Roweis' method, we could alternate between fixing  $Q$  and  $P$ , thereby obtaining a series of efficiently solvable least squares problems without requiring imputation. Each update of  $Q$  or  $P$  decreases  $\text{Err}(P, Q)$ , so the process must converge. However, we recommend using a different process for reasons that will be clarified shortly.

A major question is what would be the optimal value of  $f$ , the rank of the matrices  $Q$  and  $P$ , which represents the number of latent factors we will compute. As  $f$  grows, we have more flexibility in minimizing the squared error  $\text{Err}(P, Q)$  (16), which must decrease as  $f$  increases. However, while  $\text{Err}(P, Q)$  measures our ability to recover the known ratings, the unknown ratings are the ones that really interest us. Achieving a low error  $\text{Err}(P, Q)$  might involve overfitting the given ratings, while lowering the estimation quality on the unknown ratings. Importantly, our decision to avoid imputation leaves us with fitting a relatively low number of known entries. Therefore, the problem does not allow many degrees of freedom, preventing the use of more than a very few factors. In fact, our experience with the process showed that using more than two factors ( $f > 2$ ) degrades estimation quality. This is an undesirable situation, as we want to benefit by increasing the number of factors, thereby explaining more latent aspects of the data. However, we find that we can still treat only the known entries, but accompany the process with shrinkage to alleviate the overfitting problem.

The key to integrating shrinkage is computing the factors one by one, while shrinking the results after each additional factor is computed. This way, we increase the number of factors, while gradually limiting their strength. To be more specific, let us assume that we have already computed the first  $f - 1$  factors, which are columns

$1, \dots, f - 1$  of matrices  $P$  and  $Q$ . We provide below a detailed pseudo code for computing the next factor, which is the  $f$ -th column of matrices  $P$  and  $Q$ :

```

ComputeNextFactor(Known ratings:  $r_{ui}$ ,
                    User factors  $Q_{m \times f}$ , Item factors  $Q_{n \times f}$ )
% Compute  $f$ -th column of matrices  $P$  and  $Q$  to fit given ratings
% Columns  $1, \dots, f - 1$  of  $P$  and  $Q$  were already computed

Constants:  $\alpha = 25$ ,  $\epsilon = 10^{-4}$ 
% Compute residuals—portion not explained by previous factors
for each given rating  $r_{ui}$  do
     $\text{res}_{ui} \leftarrow r_{ui} - \sum_{l=1}^{f-1} P_{ul} Q_{il}$ 
     $\text{res}_{ui} \leftarrow \frac{n_{ui} \text{res}_{ui}}{n_{ui} + \alpha f}$  % shrinkage

% Compute the  $f$ -th factor for each user and item by solving
% many least squares problems, each with a single unknown
while Err( $P^{\text{new}}$ ,  $Q^{\text{new}}$ ) / Err( $P^{\text{old}}$ ,  $Q^{\text{old}}$ ) <  $1 - \epsilon$ 
    for each user  $u = 1, \dots, n$  do
         $P_{uf} \leftarrow \frac{\sum_{i:(u,i) \in \mathcal{K}} \text{res}_{ui} Q_{if}}{\sum_{i:(u,i) \in \mathcal{K}} Q_{if}^2}$ 
    for each item  $i = 1, \dots, m$  do
         $Q_{if} \leftarrow \frac{\sum_{u:(u,i) \in \mathcal{K}} \text{res}_{ui} P_{uf}}{\sum_{u:(u,i) \in \mathcal{K}} P_{uf}^2}$ 
return  $P, Q$ 
```

This way, we compute  $f$  factors by calling the function `ComputeNextFactor`  $f$  times, with increasing values of  $f$ . A well tuned shrinkage should insure that adding factors cannot worsen the estimation. The shrinkage we performed here,  $\text{res}_{ui} \leftarrow \frac{n_{ui} \text{res}_{ui}}{n_{ui} + \alpha f}$ , reduces the magnitude of the residual according to two elements. The first element is the number of already computed factors -  $f$ . As we compute more factors, we want them to explain lower variations of the data. The second element is the support behind the rating  $r_{ui}$ , which is denoted by  $n_{ui}$ . This support is the minimum between the number of ratings by user  $u$  and the number of users that rated item  $i$ . As the support grows, we have more information regarding the involved user and item, and hence we can exploit more factors for explaining them. By using shrinkage we observed improved estimation as factors were added. However, estimation improvement levels off beyond 30–50 factors and becomes insignificant; see Section 5. The second part of the function computes the  $f$ -th factor, by alternating between fixing item-values and user-values. We can conveniently deal with each user/item separately, thus the resulting least squares problem is trivial involving only one variable. The iterative process converges when no significant improvement of  $\text{Err}(P, Q)$  is achieved. Typically, it happens after 3–5 iterations.

At the end of the process we obtain an approximation of all ratings in the form of a matrix product  $PQ^T$ . This way, each rating  $r_{ui}$  is estimated as the inner product of the  $f$  factors that we learned for  $u$  and  $i$ , that is  $p_u^T q_i$ . A major advantage of such a regional, factorization-based approach is its computational efficiency. The computational burden lies in an offline, preprocessing step where all factors are computed. The actual, online rating prediction is done instantaneously by taking the inner product of two length- $f$  vectors. Moreover, since the factors are computed by an iterative algorithm, it is easy to adapt them to changes in the data such as addition of new ratings, users, or items. We can always train the relevant variables by running a few additional restricted iterations of the process updating only the relevant variables. While the factorization-based approach is significantly faster than our neighborhood-based approach, it is very competitive in terms of estimation quality, as we will show in Section 5. A fur-

ther improvement in estimation quality is obtained by combining the local information provided by neighborhood-based approaches, with the regional information provided by the factorization-based approach. We turn now to one way to achieve this.

#### 4.4 Neighborhood-aware factorization

The factorization-based approach describes a user  $u$  as a fixed linear combination of the  $f$  movie factors. That is, the profile of user  $u$  is captured by the vector  $p_u \in \mathbb{R}^f$ , such that his/her ratings are given by  $p_u^T Q^T$ . Interestingly, we can improve estimation quality by moving from a fixed linear combination ( $p_u$ ) to a more adaptive linear combination that changes as a function of the item  $i$  to be rated by  $u$ . In other words, when estimating  $r_{ui}$  we first compute a vector  $p_u^i \in \mathbb{R}^k$  – depending on both  $u$  and  $i$  – and then estimate  $r_{ui}$  as  $(p_u^i)^T q_i$ . The construction of  $p_u^i$  is described below.

The user vector  $p_u$  was previously computed such as to minimize, up to shrinkage, the squared error associated with  $u$ :

$$\sum_{j:(u,j) \in \mathcal{K}} (r_{uj} - p_u^T q_j)^2 \quad (17)$$

Now, if we know that the specific rating to be estimated is  $r_{ui}$ , we can tilt the squared error to overweight those items similar to  $i$ , obtaining the error function:

$$\sum_{j:(u,j) \in \mathcal{K}} s_{ij} (r_{uj} - p_u^T q_j)^2 \quad (18)$$

Recall that  $s_{ij}$  is a shrunk similarity measure between items  $i$  and  $j$ . We use here an inverse power of the Euclidean distance, but other similarity measures can be applied as well. The minimizer of error function (18) – up to shrinkage – would be  $p_u^i$ , which characterizes user  $u$  within  $i$ 's neighborhood. It is still crucial to perform a factor-by-factor computation, while shrinking during the process. Therefore, we compute the  $f$  components of  $p_u^i$  one by one, as in the following algorithm:

```

NeighborhoodAdaptiveUserFactors(Known ratings:  $r_{ui}$ , user  $u$ ,
                                    item  $i$ , item factors  $Q_{m \times f}$ )
% Compute  $f$  factors associated with user  $u$  and adapted to item  $i$ 
Const  $\alpha = 25$ 
% Initialize residuals – portion not explained by previous factors
for each given rating  $r_{uj}$  do
     $\text{res}_j \leftarrow r_{uj}$ 
    % Factor-by-factor sweep:
    for  $l = 1, \dots, f$  do
         $p_u^i[l] \leftarrow \frac{\sum_{j:(u,j) \in \mathcal{K}} s_{ij} \text{res}_j Q_{jl}}{\sum_{j:(u,j) \in \mathcal{K}} s_{ij} Q_{jl}^2}$ 
        for each given rating  $r_{uj}$  do
             $\text{res}_j \leftarrow \text{res}_j - p_u^i[l] \cdot Q_{jl}$ 
             $\text{res}_j \leftarrow \frac{n_{uj} \text{res}_j}{n_{uj} + \alpha l}$  % shrinkage
    return  $p_u^i = (p_u^i[1], p_u^i[2], \dots, p_u^i[f])^T$ 
```

This introduction of neighborhood-awareness into the regional-oriented, factorization-based method significantly improves the quality of the results, compared to neighborhood-only, or regional-only approaches (see Section 5). Moreover, typically all item-item similarities (the  $s_{ij}$  values) are precomputed and stored for quick retrieval. This enables a very quick execution of the function `NeighborhoodAdaptiveUserFactors`, which contains no iterative component. Overall running time is only slightly more than for the original factorization-based approach.

A complementary step would be to recompute the item factors by making them neighborhood-aware. That is, replacing  $q_i$  with  $q_i^u$ , which can be computed analogously to  $p_u^i$  by accounting for similarities of other users to user  $u$ . Consequently, the rating  $r_{ui}$  is estimated by  $(p_u^i)^T q_i^u$ . This results in an additional improvement in estimation accuracy. Moreover, it naturally integrates item-item similarities and user-user similarities into a single estimate, by employing item-item similarities when computing the user-factors, and user-user similarities when computing the item-factors. However, making the item factors neighborhood-aware typically requires an extensive additional computational effort, when user-user similarities are not stored due to the large number of users.

## 5. EXPERIMENTAL STUDY

We evaluated our algorithms on the Netflix data [10]. As mentioned before, this dataset is based on more than 100 million ratings of movies performed by anonymous Netflix customers. We are not aware of any publicly available CF dataset that is close to the scope and quality of this dataset. The Netflix data is currently the subject of substantial analysis, and thus is likely to become a standard benchmark for CF algorithms.

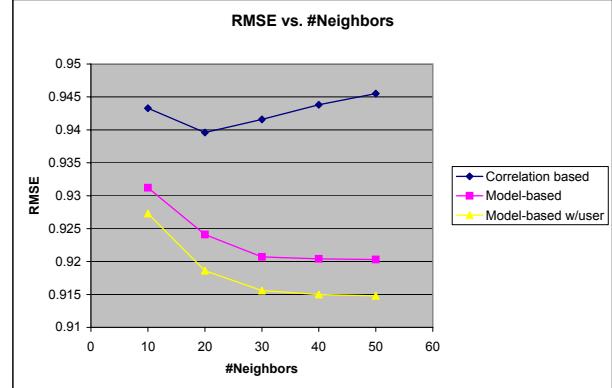
To maintain compatibility with results published by others, we adopted some standards that were set by Netflix, as follows. First, quality of the results is measured by their root mean squared error (RMSE), a measure that puts more emphasis on large errors compared with the alternative of mean absolute error. In addition, we report results on two test sets compiled by Netflix, one is known as *the Probe set* and the other is known as *the Quiz set*. These two test sets were constructed similarly, with both containing about 1.4 million of the most recent movie ratings (by date) performed by the users. The Probe set is a part of the training data and its true ratings are provided. We do not know the true ratings for the Quiz set, which are held by Netflix in order to evaluate entries. Importantly, these two test sets seem to be distributed equally across users and dates, so that the Probe data serves as a good “hold-out” sample on which to calibrate models. The test sets contain many more ratings by users that do not rate much and are harder to predict. In a way, these datasets represent real requirements from a CF system, which needs to predict new ratings from older ones, and to equally address all users, not only the heavy raters.

Netflix provides RMSE values to competitors that submit their predictions for the Quiz set. The benchmark is Netflix’s proprietary CF system, Cinematch, which achieved a RMSE of 0.9514 on this Quiz set. In the experiments that follow we focus on the Probe set, since the ratings are known. At the end of the section, we discuss results for the Quiz set. When results are reported for the Quiz set, we subsume the Probe set into our training data, which improves prediction accuracy.

### 5.1 Probe Set Results for the Local Approach

We begin with evaluating the neighborhood-based estimation discussed in Section 3. We concentrate on item-item (or, movie-movie in this case) interpolation which we found to deliver better results than the alternative user-user interpolation, confirming the findings of Sarwar et al. [15]. Our method is compared with common item-item interpolation based on Pearson correlation coefficients. While the literature suggests many flavors of correlation-based interpolation, we found that two ingredients are very beneficial here. The first is working on residuals that remain after removing global effects (Subsection 2.3). The second is working with shrunk correlations instead of the original ones (Subsection 2.2). Therefore, to make a fair comparison, we applied these two techniques to the competing correlation-based method, significantly improving

its performance. It is important to mention that our results use exactly the same neighborhoods as the correlation-based results. That is, for both methods we select as neighbors the available movies of highest shrunk Pearson correlation with the current movie. Our method differs only in calculating the interpolation weights for the neighboring ratings. In Figure 2 we compare the methods’ performance against varying neighborhood sizes ranging from 10 to 50.



**Figure 2: Comparison of three neighborhood-based approaches: a common method using shrunk Pearson correlations, our neighborhood-based approach by itself and optionally extended with user-user similarities. Performance is measured by RMSE, where lower RMSEs indicate better prediction accuracy. RMSE is shown as a function of varying neighborhood sizes on Probe set. All methods use the same neighborhood sets, differing only in the weights given to the neighbors.**

Our model-based weights consistently deliver better results than the correlation-based weights. Performance of correlation-based weights peaks at around 20 neighbors, and then starts declining as neighborhood size grows. In contrast, our methods continue to improve (at a moderating pace) as neighborhoods grow. We attribute this difference to the fact that our weights are able to reflect interdependencies among neighbors, which pairwise correlation coefficients cannot utilize. As neighborhood size grows, the number of possible interdependencies among neighbors increases quadratically; hence, more information is overlooked by correlation-based weights. For example, correlation-based approaches, or any pairwise approach for the matter, would be unforgiving to using unneeded neighbors, whose correlation is still positive (high enough to get placed among closest neighbors), whereas our model can set unneeded weights to as low as zero, if their contribution is covered by other neighbors.

An important feature of our model is the ability to integrate user-user similarities within the computation of item-item weights (Subsection 3.3). We studied the added contribution of this feature by running our model twice, once being user-aware, accounting for user-user similarities, and once without this feature. As shown in Figure 2, accounting for user similarity typically lowers the RMSE by about 0.0060; e.g., using 50 neighbors, the user-aware model results in RMSE=0.9148, versus RMSE=0.9203 without user awareness.

As indicated above, all methods were run on residuals after global effects (shrunk user mean, movie mean and various time effects) were removed. This is an important part of the modelling, but it turns out much more so for the correlation-based approach. For our method, removing these global effects was beneficial, e.g., lowering the RMSE for 50 neighbors from 0.9271 to 0.9148. However, it was more crucial for the correlation-based methods, which delivered very weak results with RMSEs above 0.99 when applied

without removing global effects. Why might this be? Before removing global effects correlations between movies are significantly higher than after removing them. Hence, we speculate that correlation coefficients, which isolate movie pairs, fail to account for strong dependencies between movies coexisting in the neighbor set. Capturing such interdependencies among neighbors is one of the major motivations for our method. However, after removing the global effects, correlations among movies become lower and hence the correlation-based methods lose far less information by isolating movie pairs. Consequently, we strongly recommend applying correlation-based methods only after removing most global effects and decorrelating items.

In terms of running time, correlation based methods possess a significant advantage over our method. After precomputing all shrunk correlations, movie-movie interpolation could predict the full Probe set in less than 5 minutes on a Pentium 4 PC. In contrast, using our more involved weights took more than a day to process the Probe set. In a newer work [2] we explain how to eliminate this running time gap.

## 5.2 Probe Set Results for Regional Approach

Now we move to the factorization approach, whose results are presented in Figure 3. The pure-regional model, which does not account for local, neighborhood-based relationships, achieved RMSEs that are slightly worse (i.e., higher) than the aforementioned movie-movie interpolation, and is actually on the same level that we could achieve from user-user interpolation. However, unlike our neighborhood-based models, the factorization-based model allows a very rapid computation, processing the full Probe set in about three minutes.

Accuracy is significantly improved by integrating neighborhood relationships into the factorization model, as explained in Subsection 4.4. This model, which integrates local and regional views on the data, outperforms all our models, achieving RMSE of 0.9090 on the Probe set. Importantly, this model is still very fast, and can complete the whole Probe set in about five minutes when using precomputed movie-movie similarities. A modest additional improvement of around 0.0015 (not shown in the figure), is achieved by integrating user-user similarities as well, but here computational effort rises significantly in the typical case where user-user similarities cannot be precomputed and stored. For these factorization-based methods the main computational effort lies in the preprocessing stage when factors are computed. This stage took about 3 hours on our Pentium 4 PC. Adding factors should improve explaining the data and thus reduce RMSE. This was our general experience, as indicated in the figure, except for a slow RMSE increase at the regional-only approach when using more than 40 factors. This probably indicates that our shrinkage mechanism was not tuned well.

## 5.3 Results on the Quiz Set

In Figure 4, we present our results for the Quiz set, the set held out by Netflix in order to evaluate entries of the Netflix Prize contest. Our final result, which yielded a RMSE of 0.8922 (6.22% improvement over Netflix Cinematch's 0.9514 result), was produced by combining the results of three different approaches: The first two are local ones, which are based on user-user and on movie-movie neighborhood interpolation. The third approach, which produces the lowest RMSE, was factorization-based enriched with local neighborhood information. Note that user-user interpolation is the weakest of the three, resulting in RMSE=0.918, which is still a 3.5% improvement over the commercial Cinematch system. The combination of the three results involved accounting for confidence scores, a topic which we briefly touch now.

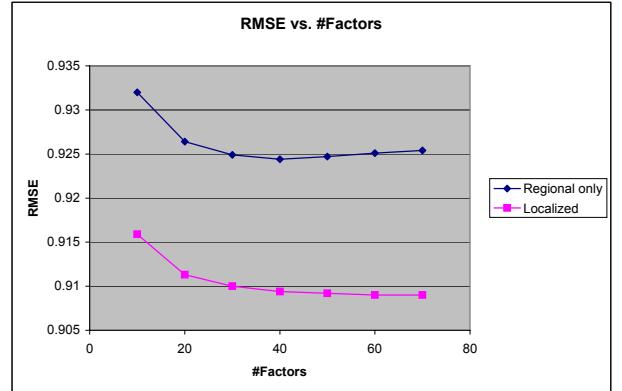


Figure 3: RMSEs of regional, factorization-based methods on Probe data, plotted against varying number of factors. The extended model, which includes also local item-item information, achieves a significant further decrease of the RMSE.

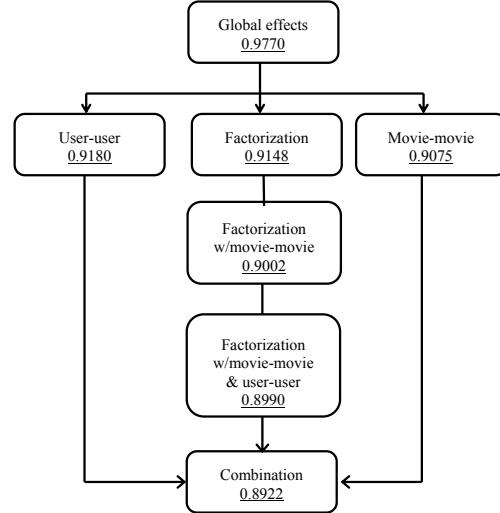


Figure 4: Our results (measured by RMSE) on the Quiz set. For comparison, the Netflix Cinematch reported result is RMSE=0.9514.

## 5.4 Confidence scores

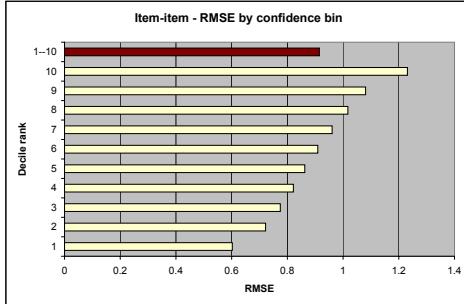
The fact that our models are based on minimizing cost functions allows us to assess the quality of each prediction, associating it with a *confidence score*. More specifically, considering the neighborhood-based methods, each score is predicted using weights that minimize problem (9). Consequently, we use the attained value of (9),  $w^T Aw / w^T Bw$ , as a suitable confidence score. Low values correspond to weights that could fit the training data well. On the other hand, higher values mean that the given weights were not very successful on the training data, and thus they are likely to perform poorly also for unknown ratings. Turning to factorization-based approaches, we can assess the quality of a user factor by (17), or by (18) in the neighborhood-aware case. Analogously, we also compute confidence scores for each movie factor. This way, when predicting  $r_{ui}$  by factorization, we accompany the rating with two confidence scores – one related to the user  $u$  factors, and the other to item  $i$  factors – or by a combination thereof.

There are two important applications to confidence scores. First, when actually recommending products, we would like to choose not just the ones with high predicted ratings, but also to account for the quality of the predictions. This is because we have more faith in predictions that are associated with low (good) confidence scores.

## Research Track Paper

This suggests a possible shrinkage of predicted ratings, to adjust for their quality. The second application of confidence scores is for joining the results of different algorithms. Different algorithms will assign varying confidence scores to each user-item pair. Therefore, when combining results, we would like to account for the confidence score associated with the results, overweighting ratings associated with better confidence scores.

Figure 5 shows how RMSEs of predicted ratings vary with confidence scores. We report results for the Probe set by using movie-movie interpolation (RMSE=0.9148). Ratings were split into deciles based on their associated confidence score. Clearly, RMSEs of deciles rise monotonically with confidence scores. Notice that top decile can be estimated quite accurately with an associated RMSE of around 0.6, while the bottom decile is much harder to estimate doubling the RMSE to above 1.2.



**Figure 5: Dependency of actual prediction accuracy (measured by RMSE) on confidence scores is revealed by partitioning Probe results into ten deciles according to confidence scores**

## 6. DISCUSSION

In this work, we designed new collaborative filtering methods based on models that strive to minimize quadratic errors, and demonstrated strong performance on a large, real-world dataset. The following components of our method were all crucial to its good performance: 1) removing “global” components up front and focusing our modelling on the residuals, 2) using shrinkage to prevent overfitting of the large number of parameters, 3) incorporating interactions among the users and movies by fitting a model that jointly optimizes parameter estimates, and 4) incorporating local, neighborhood based analysis into a regional, factorization model.

Formulating the methods around formal models allowed us to better understand their properties and to introduce controllable modifications to the models for evaluating possible extensions. Importantly, we have found that extending the models to combine multiple facets of the data, such as user-similarity with item-similarity or local-features with higher-scale features, is a key component in improving prediction accuracy. Another benefit of our error-based model is the natural definition of confidence scores that accompany the computed ratings and essentially “predict the quality of the predictions”. In a future work, we would like to study the integration of these confidence scores with the predicted ratings in order to provide better recommendation to the user. Other future research ideas include non-linear extensions to our linear models, especially to the factorization model which might be extended by such non-linear functions.

We greatly benefited from the recently introduced Netflix data, which opens new opportunities to the design and evaluation of CF algorithms. None of our models use any information about the content (actors, genres, etc) and it would be interesting to devise models to incorporate such data into our CF-based model. Since our factorization attempts to indirectly model this external information, we’d like to see how much benefit that data would provide.

## 7. REFERENCES

- [1] G. Adomavicius and A. Tuzhilin, “Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions”, *IEEE Transactions on Knowledge and Data Engineering* **17** (2005), 634–749.
- [2] R. Bell and Y. Koren, “Improved Neighborhood-based Collaborative Filtering”, submitted, 2007.
- [3] S. Deerwester, S. Dumais, G. W. Furnas, T. K. Landauer and R. Harshman, “Indexing by Latent Semantic Analysis”, *Journal of the Society for Information Science* **41** (1990), 391–407.
- [4] D. Goldberg, D. Nichols, B. M. Oki and D. Terry, “Using Collaborative Filtering to Weave an Information Tapestry”, *Communications of the ACM* **35** (1992), 61–70.
- [5] K. Goldberg, T. Roeder, D. Gupta and C. Perkins, “Eigentaste: A Constant Time Collaborative Filtering Algorithm”, *Information Retrieval* **4** (2001), 133–151.
- [6] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, 1996.
- [7] J. L. Herlocker, J. A. Konstan, A. Borchers and John Riedl, “An Algorithmic Framework for Performing Collaborative Filtering”, *Proc. 22nd ACM SIGIR Conference on Information Retrieval*, pp. 230–237, 1999.
- [8] D. Kim and B. Yum, “Collaborative Filtering Based on Iterative Principal Component Analysis”, *Expert Systems with Applications* **28** (2005), 823–830.
- [9] J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon and J. Riedl, “GroupLens: Applying Collaborative Filtering to Usenet News”, *Communications of the ACM* **40** (1997), 77–87, [www.grouplens.org](http://www.grouplens.org).
- [10] Netflix prize - [www.netflixprize.com](http://www.netflixprize.com).
- [11] G. Linden, B. Smith and J. York, “Amazon.com Recommendations: Item-to-item Collaborative Filtering”, *IEEE Internet Computing* **7** (2003), 76–80.
- [12] J. Nocedal and S. Wright, *Numerical Optimization*, Springer (1999).
- [13] S. Roweis, “EM Algorithms for PCA and SPCA”, *Advances in Neural Information Processing Systems 10*, pp. 626–632, 1997.
- [14] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, “Application of Dimensionality Reduction in Recommender System – A Case Study”, *WEBKDD’2000*.
- [15] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, “Item-based Collaborative Filtering Recommendation Algorithms”, *Proc. 10th International Conference on the World Wide Web*, pp. 285–295, 2001.
- [16] R. Tibshirani, “Regression Shrinkage and Selection via the Lasso”, *Journal of the Royal Statistical Society B* **58** (1996).
- [17] J. Wang, A. P. de Vries and M. J. T. Reinders, “Unifying User-based and Item-based Collaborative Filtering Approaches by Similarity Fusion”, *Proc. 29th ACM SIGIR Conference on Information Retrieval*, pp. 501–508, 2006.

# The Influence Limiter: Provably Manipulation-Resistant Recommender Systems

Paul Resnick  
University of Michigan  
School of Information  
presnick@umich.edu

Rahul Sami  
University of Michigan  
School of Information  
rsami@umich.edu

## ABSTRACT

An attacker can draw attention to items that don't deserve that attention by manipulating recommender systems. We describe an influence-limiting algorithm that can turn existing recommender systems into manipulation-resistant systems. Honest reporting is the optimal strategy for raters who wish to maximize their influence. If an attacker can create only a bounded number of shills, the attacker can mislead only a small amount. However, the system eventually makes full use of information from honest, informative raters. We describe both the influence limits and the information loss incurred due to those limits in terms of information-theoretic concepts of loss functions and entropies.

## Categories and Subject Descriptors

I.2.6 [Computing Methodologies]: Artificial Intelligence—*Learning*

## General Terms

Algorithms, Reliability

## Keywords

Recommender systems, manipulation-resistance, shilling

## 1. INTRODUCTION

Content posted on the Internet is not of uniform quality, nor is it equally interesting to different audiences. Recommender systems guide people to items they are likely to like, based on their own and other people's subjective reactions. We will refer to people's opinions generically as ratings, whether users explicitly enter them in the form of ratings or tags, or whether the system infers them from implicit behavioral indicators such as purchases, read times, bookmarks, or links.

Authors and other parties often want to direct attention to particular items. Google, Yahoo!, and others channel this

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

RecSys '07, October 19–20, 2007, Minneapolis, Minnesota, USA.  
Copyright 2007 ACM 978-1-59593-730-8/07/0010 ...\$5.00.

into a multi-billion dollar advertising marketplace. But to the extent that people rely on recommender systems of various kinds to guide their attention, there are also natural incentives for promoters to manipulate the recommendations. An attacker may rate strategically rather than honestly and may introduce multiple entities, sometimes called *sybils*, to rate on behalf of the attacker.

We offer a manipulation-resistance algorithm, called the Influence Limiter, that can be overlaid on existing recommender algorithms. Consider the predictions about whether a particular target person will like various items. Each rater begins with a very low but non-zero reputation score. The current reputation limits the influence she<sup>1</sup> can have on the prediction for the next item. Eventually, the target person indicates whether he likes the item and the raters who contributed to predicting whether the target would like it gain or lose reputation. The more that a rating implies a *change* in the prediction for the target, the greater the potential change in the rater's reputation score. A rater who simply goes along with the previous change will have no impact and thus get no change in her reputation.

The Influence Limiter has several desirable properties. First, in order to maximize the expected reputation score of a single rater endowed with some information about the target's likely response to the items, the optimal strategy is to induce predictions that accurately reveal that rater's information about the items. If the underlying recommendation algorithm is making optimal use of ratings, this implies that entering honest ratings is optimal. An important special case is that a rater who has not interacted with an item, and therefore has no information about the target's likely response to it, can only lose reputation in expectation by giving a rating.

Second, the actual reputation score of any rater is always positive and is bounded above by an information-theoretic measure of the actual improvement that rater has made in the predictions for the target. It is not possible to prevent all manipulations of the predictions for particular items—a rater who provides good information on all other items but strategically provides bad information on one item is indistinguishable from a rater who simply has an unusual opinion on the item in question. Our algorithm does, however, ensure that no rater can negatively impact the overall set of recommendations for a target by more than a tiny amount. Moreover, if the item being manipulated is unlikely to be of interest to the target, later raters may provide information

<sup>1</sup>We refer generically to raters as female and the target for predictions as male.

that corrects the prediction on that item before the target is affected by the recommendation.

Finally, our algorithm limits the amount of damage that can be done with sybils. For example, if one rater provides bad information about an item in order to increase the reputation of another rater who later corrects that bad information, the expected sum of the reputations of the two raters does not increase. Thus, while it may be possible to transfer reputation among sybils, it is not possible to increase the total reputation of the raters that a person controls. We presume that the recommender system imposes some minimal cost (or inconvenience) on the creation of rater entities. Thus, there is some bound (say, 1,000) on the number of sybils one person can create without it being too costly and without being detected. The initial reputation of each rater is set low enough that the total reputation of this bounded number of raters is still relatively small.

To further motivate our manipulation-resistance algorithm, consider some approaches to manipulating conventional recommender systems. One threat is a *cloning* attack. For example, in a recommender system that asks each rater to report movie ratings on a 1-5 scale, the attacker simply reports the same ratings as some other rater, except for a single item to be manipulated. Most recommender systems do not take into account the order of ratings, and thus the attacker will have just as much effect on predictions for the last item as the rater who was copied. In a nearest-neighbor recommender algorithm, the attacker can even just clone the ratings of the target; the attacker will then be the nearest neighbor of the target. The cloning attack can be made more difficult by hiding the actual rating vectors of raters, but significant information about others' ratings will leak out in the content of the recommendations, and sophisticated attackers will be able to create influential rating profiles through approximate cloning. Our approach thwarts the cloning attack by adding reputation only when a rater *improves* the prediction made for some target. Unless the rater moves the recommendation from where it was before the rater provided its information, the rater can neither gain nor lose reputation.

A second threat to conventional recommender systems comes from *random profile flooding*. An attacker creates a large number of sybils that provide random reports except on the item or items to be manipulated. By chance, some sybils may appear to have provided useful information in their random reports. These sybils are used to impact the prediction for an item being manipulated. Our algorithm thwarts this attack by making the probability of gaining sufficient credibility through random reports very low, so low that an attacker gains less influence in expectation from its sybils making random guesses than from simply transferring the initial credibility of all its sybils to a main identity. Moreover, any sybil profile that does happen to gain a high credibility score has, by chance, moved the predictions for the target in a useful way, thus compensating for the lost utility from the subsequent manipulation.

The paper begins with an exploration of related work in section 1.1. Section 2 presents a model of the recommending process. Section 3 presents our algorithm. Section 4 provides formal statements of its manipulation-resistance properties. Section 5 presents information-theoretic bounds on the information loss due to influence limits. Section 6 discusses limitations and possible extensions.

## 1.1 Related Work

The possibility of sybil attacks on recommendation systems has been noted by O'Mahony *et al.* [22] and Lam and Riedl [15], who use the term "shilling attack". Through simulation, Lam and Riedl study versions of the cloning and random profile attacks on different recommender algorithms, and note that the effectiveness of the attack varies depending on the algorithm used. However, they do not address the development of a provably attack-resistant algorithm.

Several authors have suggested using statistical metrics on ratings to distinguish "attack" identities from "regular" identities, and eliminate the former [7, 23, 18]. Mobasher *et al.* [20] survey this literature and classify attack strategies. This approach is likely to lead to an arms-race where shillers employ increasingly sophisticated patterns of attack. To avoid this, our approach does not rely on identifying particular attack identities or specific attack strategies. O'Donovan and Smyth [21] suggest using accuracy information from multiple targets to judge credibility; it would be interesting to see if our scheme can be extended in this way.

Dellarocas [8] provides an algorithm that bounds the damage that attackers can do when they collectively provide less than half the ratings in the system and the honest ratings are normally distributed. Our approach succeeds much more generally, even in situations where only a tiny fraction of the ratings are honest, at the expense of greater information loss during the startup phase when raters are not yet credible.

Herlocker *et al.* [13] study a modification of a nearest-neighbor recommender algorithm that does not count a rater as a near neighbor until it has rated sufficiently many items in common with a target rater. This is similar in spirit to our influence-limiting approach, but we provide a limiting process that is grounded in information theory and provably resistant to manipulation.

We use proper scoring rules to elicit honest ratings. These were pioneered in the context of forecasting objective events like weather patterns [4]; Miller *et al.* [19] noted that scoring rules could be adapted to the recommendation setting by treating the target's rating as an objective outcome. Hanson [11] developed the *market scoring rule* as a mechanism for information markets. In this mechanism, a trader is rewarded with the score difference between her prediction and the previous prediction. This relative scoring rewards the first provider of information, and forms an essential component of our approach. We develop additional machinery to handle strategies involving sybils and bankruptcy.

Bhattacharjee and Goel [3] suggest sharing the revenue of a ranking system with the raters. Using techniques similar to market scoring rules to determine the revenue shares, they argue that an attack on the system would be costly. In contrast, we do not require any real money transactions, and prove bounds on the damage that sybils can do.

We use the error score change as a natural measure of performance of the system and damage to the system. Rashid *et al.* [25] propose several other algorithm-independent measures of rater influence; unlike error score change, their measures do not consider the dynamic order of ratings. Separately, Rashid *et al.* [24] use entropy to analyze a different problem: choosing which items to ask new users to rate.

The literature on bounded-regret online learning deals with combining predictions from multiple forecasters and proving worst-case bounds on the error relative to the best predictor that could be chosen in hindsight (see Cesa-Bianchi

and Lugosi [5] and references therein). Many online learning algorithms can also be viewed as schemes for betting on a sequence of events [26]. The influence-limiting algorithm can be interpreted as an online learning algorithm: our damage bound is a form of relative error bound. The online learning literature typically does not take the order of forecasts on a single item into account; our algorithm is tailored to this critical feature of the recommender setting.

Awerbuch *et al.* [2] study manipulation in a different model of the recommendation process: a user samples items and recommendations until he likes an item, at which point he recommends that single item to others. They present a sampling algorithm and prove bounds on the number of samples required, even in the presence of adversaries. Awerbuch and Kleinberg [1] describe an online learning scheme that is provably good for a generalization of this problem that incorporates time-varying preferences and recommendations.

Sybilproofness has been studied in the context of reputation systems by Cheng and Friedman [6]. Their mechanisms begin with a trust graph: expressions of trust by raters about *other raters*. Advogato [16] and Eigentrust [14] also attempt to address manipulation in a trust-graph model. Massa and Bhattacharjee [17] show how external trust relationships can be used to improve recommender systems. These techniques are not directly usable in our setting, because raters may not know anything about the other raters in the system. In fact, our algorithm can be viewed as a way to securely derive a trust (or credibility) graph from ratings on inanimate objects.

## 2. THE RECOMMENDING PROCESS

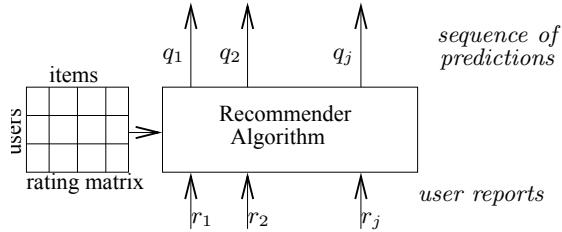


Figure 1: A recommender system

We now present a formal model of the recommending process, illustrated in Figure 1. Let  $N$  denote the total number of identities in the system, and  $\mathcal{N}$  denote the set of identities. We use  $M$  to denote an upper bound on the number of items that are available for rating, so that each item can be assigned an index in  $\{1, \dots, M\}$ .

There is a space  $\mathcal{L}$  of possible labels, which describes the classification of items by the target for whom the system makes predictions. The prediction space  $\mathcal{X}$  is a set of probability distributions over the label space; a prediction  $\mathbf{q} \in \mathcal{X}$  is thus a distribution over labels. Prior to the target providing a label, a sequence of raters provide ratings  $r_1, r_2, \dots$  that change the recommender system's assessment of the target's probability distribution over labels.

We describe the process assuming that all raters enter ratings into the same underlying recommender algorithm, which combines them with previous ratings to generate predictions; this matches the standard mode of operation of current recommender systems. The formal analysis extends

to more general settings: raters could directly report probabilities, incorporating past information in any way they like. In practice, the recommender algorithm would also use each incoming rating to update predictions about other items and for more than just one target person. It is sufficient for our purposes to describe the predictions for a single target.

Existing recommenders typically predict a single number, which corresponds to the mean of the distribution over labels on a numeric scale (e.g., 3.8 on a 1-5 scale). To function in our scheme, such recommenders could be extended to report entire probability distributions (e.g., 30% chance of 5; 20% chance of 4; 50% chance of 3). Alternatively, intermediate labels and point predictions can both be interpreted as a mixture of ratings over the extreme points (e.g., 3.8 is a 70% chance of 5 and 30% chance of 1). However, in the special case where there are only two possible labels, the mean (e.g., .7 on a 0-1 scale) completely determines the entire distribution. In the rest of this paper, we shall assume that the label space is simply  $\{HI, LO\}$ , and that a prediction expresses the probability of a *HI* label.

### 2.1 Measuring Error: Loss Functions

In this section, we describe the error measures we use for a single prediction. Later, in section 5.1, we will see that these error measures lead naturally to information-theoretic measures of a rater's expected contribution.

We begin by assigning a value to good recommendations. We do this by postulating that the target has a *loss function*  $L(l, q)$ , where  $q \in [0, 1]$  is the recommendation (predicted probability of rating *HI*) that the target was given, and  $l \in \{HI, LO\}$  is the target's ultimate label. One common choice is the *log-loss* function:

$$L(HI, q) = -\log q; \quad L(LO, q) = -\log(1 - q)$$

Although typically logarithms to base 2 are used, so as to measure entropy in bits, we use natural logarithms (logarithms to base  $e$ ) throughout this paper. Note that the loss is 0 when the prediction is completely accurate (i.e., when there is no error in the prediction).

One drawback of the log-loss function is that it is unbounded as  $q$  tends to 0 or 1. We need to avoid the possibility of infinite (positive or negative) scores. One alternative is the *Quadratic loss function/scoring rule*:

$$L(HI, q) = (1 - q)^2; \quad L(LO, q) = q^2$$

This corresponds to using the expected squared error (the variance) as a measure of uncertainty, which has a long history in statistics. It also corresponds to the use of mean squared error as a measure of prediction accuracy in recommender systems that make predictions on a 0-1 scale. The quadratic loss function is clearly bounded by  $[0, 1]$ .

## 3. THE INFLUENCE LIMITER ALGORITHM

Figure 2 depicts an influence-limited recommender system. There are two key additional features. First, the prediction  $q_j$  output by the recommender algorithm passes through an influence-limiting process to produce a modified prediction  $\tilde{q}_j$ . The influence-limiting process generates  $\tilde{q}_j$  as a weighted average of the prediction  $\tilde{q}_{j-1}$  prior to incorporating  $j$ 's rating and the prediction  $q_j$  using  $j$ 's rating. The weighting of the two terms depends on the reputation  $R_j$  that  $j$  has accumulated with respect to the target. When  $R_j \geq 1$ , all weight is on  $q_j$ , i.e.,  $j$  has full credibility.

The second feature is a scoring function that assigns reputation to  $j$  based on whether or not the target actually likes the item. The amount of reputation that can be gained is again limited by the current reputation, and is selected so that the reputation will always be positive. It is also tuned so that honest raters reach full credibility after  $O(\log n)$  ratings, to limit the amount information that is discarded in the influence-limiting step before a rater reaches full credibility. If a rater's entire reputation was risked on each rating, the probability of getting to full credibility from a sequence of ratings the target agrees with would be too small and the expected time to full credibility would be too high. The rate limiter  $\beta_j$  threads the needle, allowing the rater to risk more as her reputation rises, but not risk too much, as we prove in section 5.<sup>2</sup>

Figure 3 presents the algorithm more formally. Each rater begins with a tiny reputation  $e^{-\lambda}$ , small enough so that even a large number of raters would have total reputation much less than 1.  $\beta_j = \min(1, R_j)$  determines  $j$ 's influence limit; if  $R_j$  is above 1,  $j$  can move the prediction  $\tilde{q}_{j-1}$  to  $q_j$ , but if  $R_j$  is below 1 she can only move it partway there. The score for rater  $j$  on the current item is a fraction  $\beta_j$  times the market scoring rule, the difference between the loss function for the prediction  $\tilde{q}_{j-1}$  and the loss function for  $q_j$ . Note that she is scored on the prediction she would have liked to make,  $q_j$ , even if the rate limiter only permitted her to move the prediction to  $\tilde{q}_j$ .

The log loss and quadratic loss functions are *proper scoring rules* [4, 9]. That means that its expected value is maximized when  $q_j$  matches the true probability that the target likes the item. This eliminates any incentive for rater  $j$  to lie about her rating, if she wants to maximize her expected reputation score, a property that we prove more formally in section 4. Note that honest reporting only maximizes the score in expectation; given the target's true opinion of the item, the actual loss is minimized with a prediction of that outcome with certainty.

## 4. STRATEGIC PROPERTIES

Our main non-manipulation result shows that, for any attack strategy involving up to  $n$  sybils, the net negative impact (damage) due to the attacker is bounded by a small amount. We first state two important assumptions we make about the attack strategy, and then prove the results on damage limits. We also show that a user seeking to maximize her influence has a strict incentive to rate honestly.

**Assumptions:** One important assumption we make is that there is a number  $n$  that bounds the maximum number of fake identities a single attacker can create. As stated in section 1, we intend  $n$  to be a fairly large number, perhaps in the thousands or even millions. It might appear that limiting a user to a million identities is qualitatively as difficult as limiting her to one, but there is an important difference in practice. For example, many sites require new users to solve CAPTCHAs [27], which are puzzles that require a few seconds of human attention. While this is a mild inconvenience for most users who create only one account (or

<sup>2</sup>By analogy, consider a sequence of coin flips where you double your money on heads and give back all but \$1 on tails. The expected time to reach a bankroll of  $x$  would be quite long, even if heads occurred with probability 0.75. By contrast, if tails only required giving back half your money, the expected time would be  $2 \log x$ .

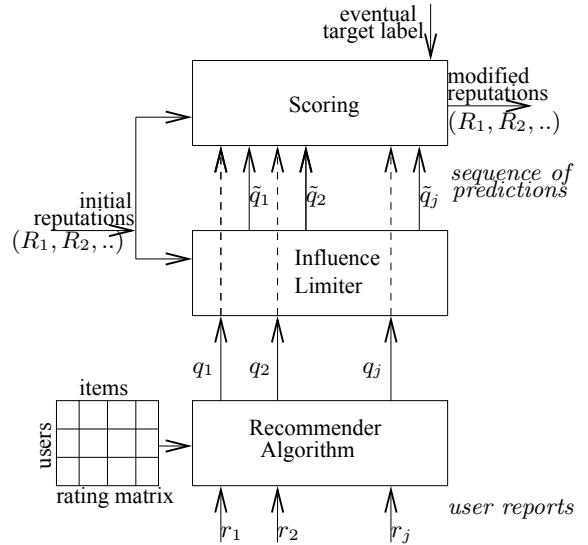


Figure 2: An influence-limited recommender system

ComputeReputations( $\lambda$ )

1. Initialize  $R_j = e^{-\lambda}$  for all  $j$ .
2. For an item the target will eventually label do:
  - a.  $\tilde{q}_0 = p_0$
  - b. Consider the ratings on the item in temporal order
  - c. For each rater  $j$ :
  - d.  $\beta_j = \min(1, R_j)$
  - e.  $\tilde{q}_j = (1 - \beta_j)\tilde{q}_{j-1} + \beta_j q_j$
  - f. After the target provides label  $l$ ,  $R_j = R_j + \beta_j [L(l, \tilde{q}_{j-1}) - L(l, q_j)]$

Figure 3: Algorithm to compute reputations and limited predictions. Loss function  $L$  can be any proper scoring rule bounded by  $[0, 1]$ .

a few accounts), it can make creating millions of accounts prohibitively expensive. On the other hand, the only realistic way to limit a user to only one account is to use and verify personally identifying information for each account, which is a major privacy threat, and can be costly and time-consuming. Note that we do not assume that a constant fraction of the identities in the system are real users; the set of identities can be dominated by an attacker's sybils.

Secondly, we restrict our attention throughout this paper to *myopic* attack strategies. In particular, we do not consider reputation-building strategies in which the attacker enters poor ratings that mislead later raters into amplifying her misinformation, and then later corrects it with an additional rating from another sybil. This is a nontrivial assumption: In many cases, the optimal prediction after honest ratings from agents 1, 2, ..,  $j$  will be sensitive to each of their inputs and might amplify the change in predictions by earlier raters.

These non-myopic strategies, if they exist, would be equivalent to manipulative strategies in information markets, analogous to initiating a buying frenzy for a particular stock and then selling to the very buyers who entered the market because of your actions. While there are market settings in which non-myopic strategies are theoretically possible, several studies have shown that such manipulation attempts are not very successful in practice (see, e.g. [12]). Non-myopic manipulative strategies, if they exist, are also likely to be quite complex for an attacker to carry out, as they would require delicate calculations about other agents' inference and learning methods. In this paper, we restrict our attention to preventing the simpler (but still very powerful) myopic attacks. Research on non-myopic manipulation in information markets, and methods to guard against them, will have direct relevance to our mechanism.

**Impact of rater  $j$ :** We now introduce a measure of rater  $j$ 's impact on the recommendation quality.  $j$ 's rating  $q_j$  on an item  $i$  has an immediate impact on the recommendations: The recommendation  $\tilde{q}_{j-1}$  is changed to the recommendation  $\tilde{q}_j$ . We define the *myopic impact attributed to  $j$  for item  $i$* ,  $\Delta_j^i$ , by:

$$\Delta_j^i \stackrel{\text{def}}{=} L(l^i, \tilde{q}_{j-1}) - L(l^i, \tilde{q}_j)$$

(Here,  $l^i$  is the target's rating on item  $i$ .) In other words,  $\Delta_j^i$  measures the reduction in prediction loss that resulted from the immediate changes in recommendation following  $j$ 's rating.

Note that myopic impacts are additive. The total impact of a sequence of raters is the reduction in error from the initial default prediction to the final prediction  $\tilde{q}_J$ . This can be expressed as the sum of the impacts attributed to the individual raters because the terms telescope:

$$\sum_j \Delta_j^i = \sum_j [L(l^i, \tilde{q}_{j-1}) - L(l^i, \tilde{q}_j)] = L(l^i, p_0) - L(l^i, \tilde{q}_J)$$

Finally, we define the *net impact* of rater  $j$  by the sum of her myopic impacts on all items  $i$ :  $\Delta_j = \sum_i \Delta_j^i$ .

We now prove our non-manipulation results in this context. First, we make a straightforward but important observation about the reputations calculated by the algorithm:

*Observation:*  $R_j \geq 0$

This is true initially, and it is clearly maintained in step 2.f of the algorithm because  $L()$  is bounded between 0 and 1.

**LEMMA 1. (Honest Reporting)** *For any entity  $j$ , if  $j$  believes (at the time of rating) that the probability of the item being labelled HI is  $q$ ,  $j$  optimizes her expected reputation gain on this item by putting in a rating such that  $q_j = q$ .*

**PROOF.** From line 2.d, it can be seen that the change  $\Delta R_j$  in  $j$ 's reputation is proportional to  $L(l_i, \tilde{q}_{j-1}) - L(l_i, q_j)$ . Agent  $j$  cannot control the first term. As the scoring rule is proper, the second term is minimized in expectation by reporting  $q_j = q$ .  $\square$

*Remark:* In itself, this is an incentive for honest rating only if users care about their reputations or influence. Inducing users to care about reputations is an orthogonal issue; this might be done through status rewards for raters with high reputation, or by adding a little noise to the recommendations for users with low reputation.

*Remark:* If the recommender algorithm that generates the predictions  $q_j$  from the ratings  $r_1, \dots, r_J$  is imperfect, rater

$j$  may benefit by compensating for the algorithm: reporting a value  $r_j$  different from what she truly perceived in order to improve the prediction.

The following critical lemma relates the influence an agent garners to her measured impact on prediction error.

**LEMMA 2. (Reward-performance inequality)** *The reputation increase  $\Delta R_j$  of player  $j$  on an item  $i$  is no more than the impact  $\Delta_j^i$  of entity  $j$  on this item's recommendation.*

**PROOF.** Let  $\beta_j$  denote the influence limit calculated in step 2.d of the algorithm and  $l$  denote the eventual label on this item. Then, the reputation earned by player  $j$  for this item is

$$\Delta R_j = \beta_j [L(l, \tilde{q}_{j-1}) - L(l, q_j)]$$

The reduction in prediction error, on the other hand, is given by

$$\begin{aligned} \Delta_j^i &= L(l, \tilde{q}_{j-1}) - L(l, \tilde{q}_j) \\ &= L(l, \tilde{q}_{j-1}) - L(l, (1 - \beta_j)\tilde{q}_{j-1} + \beta_j q_j) \end{aligned}$$

(Note that  $\Delta R_j$  and  $\Delta_j^i$  may be negative.) For both scoring rules we consider, log and quadratic, the loss functions are convex, and so we have:

$$L(l, (1 - \beta_j)\tilde{q}_{j-1} + \beta_j q_j) \leq (1 - \beta_j)L(l, \tilde{q}_{j-1}) + \beta_j L(l, q_j)$$

Thus, substituting in the expression for  $\Delta_j^i$ , we get

$$\Delta_j^i \geq \beta_j [L(l, \tilde{q}_{j-1}) - L(l, q_j)] = \Delta R_j \quad \square$$

**COROLLARY 3.** *At any point of time, the total myopic impact  $\Delta_j$  of all ratings by entity  $j$  satisfies  $\Delta_j \geq -e^{-\lambda}$ .*

**PROOF.** From the algorithm, it is clear that  $\beta_j$  and  $R_j$  are never negative. Thus, the net decrease in reputation of entity  $j$  is at most  $e^{-\lambda}$ . By lemma 2, the net increase in prediction error  $\Delta_j = \sum \Delta_j^i$  due to  $j$ 's ratings is no more than this.  $\square$

We can now state our main manipulation resistance property:

**THEOREM 4. (Limited Damage)** *Consider any strategy for the attacker  $k$ ; the strategy involves the creation of up to  $n$  sybils  $K = \{k_1, k_2, \dots, k_n\}$ . Then, the total myopic impact of all of  $k$ 's sybils is bounded by*

$$\sum_{k_t} \Delta_{k_t} \geq \sum_{k_t} R_{k_t} - ne^{-\lambda} \geq -ne^{-\lambda}$$

**PROOF.** It follows from Corollary 3 by simply summing over all sybil identities.  $\square$

*Remark:* Setting  $\lambda = \log(cn)$  for some parameter  $c$ , Theorem 4 shows that an attacker cannot cause a total reduction in the system performance of more than  $1/c$ . This damage bound does not require any assumptions about rationality, prior probabilities, or honest ratings.

Note, however, that we are relying on the assumption that strategies that involve misleading other raters are impossible to carry out: otherwise,  $\Delta_j$  would not completely capture the effect  $j$  has had on the recommendations. If one sybil can induce subsequent raters to amplify its effect on the predicted probability, then a later sybil may be able to secure an inflated  $\Delta$  score, by correcting the amplified misinformation. The sum of all impacts still correctly measure the total reduction in prediction but the attacker would be stealing some impact score from the other raters who were misled by the initial sybil.

## 5. BOUNDING THE INFORMATION LOSS

Manipulation-resistance in itself is not very hard to achieve; simply ignoring all user ratings would result in nonmanipulable recommendations. It is therefore necessary to show that the recommendation scheme is still *informative*. In particular, the influence-limited operation does not make full use of the information from a user with low reputation. In this section, we show that users will in expectation require  $O(\lambda) = O(\log n)$  ratings to build up their influence to an adequate level. We use this to bound the total information loss. We work primarily with the quadratic loss function, but the results should generalize to other loss functions with different constants.

### 5.1 Partial Information Model

In order to analyze the performance of the system in terms of reducing uncertainty, we need a formal model that captures the uncertainty of rating, raters' partial information signals, and raters' ability to learn from (and copy) previous ratings. We use a standard model of information partitions, which is described below. This model is used purely to analyze information loss; the operation of the Influence Limiter algorithm does not depend on it.

There is a space  $\Omega$  of possible states (item types). For example, each state in  $\Omega$  might correspond to a particular combination of a large number of movie attributes. For each  $\omega \in \Omega$ , there is a corresponding value  $l(\omega) \in \{HI, LO\}$  that describes whether the target will like items in that state (*i.e.*, with that combination of attributes). Further, we assume there is a common prior probability distribution  $p : \Omega \rightarrow [0, 1]$  that defines the relative likelihood of different states in the absence of additional information. We use  $p_0$  to denote  $P_p(l(\omega) = HI)$ , the prior probability, before taking into account any ratings, that a randomly chosen item will be labelled *HI* by the target. We assume that the state space is finite.

When a rater acquires a signal about an item, we model her as eliminating some of the possible states, but not changing the relative likelihood of the remaining states. For example, she might be able to tell if the movie is violent or not, but not be able to discern some other characteristic, such as humor, that would distinguish among violent movies. More formally, all information acquisition is modelled in terms of identifying a component of a partition  $\pi$  of the state space  $\Omega$ . A rater  $j$  has a partition  $\pi_j = \{\pi_j^1, \dots, \pi_j^t\}$ . This is a partition, so  $\pi_j^s \cap \pi_j^k = \emptyset$  for  $s \neq k$ , and  $\cup_s \pi_j^s = \Omega$ . The interpretation is that after acquiring a signal about the item, e.g., by watching the movie,  $j$  will know which component  $\pi_j^s$  of her partition the true state belongs in, but will not be able to distinguish two states in the same partition.

Given the loss function  $L$ , a natural way to frame the objective of the recommender system is to attempt to minimize the total error in the predictions, as measured by the sum of the losses. If no partial information from raters was available, the best prediction the system (or the target herself) could make would be to say  $q = p_0 = P_p(l(\omega) = HI)$ . This value of  $q$  can be shown to minimize the expected loss in the absence of any other information; the expected loss on each item is then given by the *entropy*  $H(l)$ :  $H(l) = -p_0 L(HI, p_0) + (1 - p_0)L(LO, (1 - p_0))$ .

The log-loss function (or *logarithmic scoring rule*) allows us to frame our results in terms of standard information-theoretic entropy. Other loss functions can be used as scor-

ing rules in our algorithm, and would correspond to other measures of entropy, relative entropy, etc. (see the article by Grunwald and Dawid [10] for more information). The entropy measures thus derived are conceptually similar to the standard entropy.

Now, consider a sequence of raters  $1, 2, \dots, j$ ; each rating identifies a component of that rater's partition. Then, the prediction made after  $j$ 's rating will depend on  $j$ 's rating as well as all previous ratings. The information available to the recommender algorithm can be represented by a partition  $\hat{\pi}_j$  that is a refinement of  $\pi_j$ , reflecting the fact that it can distinguish between items which  $j$  rated differently as well as possibly some items which were in the same component of  $\pi_j$ , but different components of the partition of some  $\pi_k$  for  $k < j$ . An ideal recommender algorithm then makes the best possible prediction  $q_j$  given this information.

A sequence of ratings by raters  $1, 2, \dots, j$  thus defines a sequence of partitions  $\hat{\pi}_1, \dots, \hat{\pi}_{j-1}, \hat{\pi}_j$ . Generally,  $\hat{\pi}_j$  is a *refinement* of  $\hat{\pi}_{j-1}$ : no component of  $\hat{\pi}_j$  overlaps multiple components of  $\hat{\pi}_{j-1}$ .<sup>3</sup>

We can now define the innate *informativeness* of player  $j$  as the expected reduction in loss due to player  $j$ 's participation as the  $j$ th rater in the sequence, before we learn what any of the realized ratings are. For any component  $s \in \hat{\pi}_j$ , define  $s_{j-1} \in \hat{\pi}_{j-1}$  as the component of  $\hat{\pi}_{j-1}$  containing  $s$ . We can then define  $q_j(s) = P(l(\omega) = HI | \omega \in s)$  and  $q_{j-1}(s) = P(l(\omega) = HI | \omega \in s_{j-1})$ . That is,  $q_j(s)$  is the posterior probability the target will like the item if the state is in the partition  $s$ , and  $q_{j-1}(s)$  is the predicted probability after the rating sequence that would occur when the state is in  $s$ , but before taking into account  $j$ 's rating.

The informativeness (*i.e.*, expected error reduction)  $I(\hat{\pi}_j || \hat{\pi}_{j-1})$  due to the addition of user  $j$ 's private information is calculated as:

$$\begin{aligned} I(\hat{\pi}_j || \hat{\pi}_{j-1}) &\stackrel{\text{def}}{=} \sum_{\omega \in \Omega} p_\omega [L(l(\omega), q_{j-1}(\omega)) - L(l(\omega), q_j(\omega))] \\ &= \sum_{s \in \hat{\pi}_j} p_s D(q_j(s) || q_{j-1}(s)) \end{aligned}$$

where  $D(q || r) = q[L(HI, r) - L(HI, q)] + (1 - q)[L(LO, r) - L(LO, q)]$  is the *relative entropy*, a central construct in information theory. We can formally extend this definition to define  $I(\mathbf{q} || \mathbf{u})$  for any functions  $\mathbf{q}(\omega)$  and  $\mathbf{u}(\omega)$  that are constant on components  $s$ , such as the influence-limited predictions  $\tilde{q}_j$ .

This  $I(\hat{\pi}_j || \hat{\pi}_{j-1})$  is our measure of the incremental value of  $j$ 's information given the ordering  $1, 2, \dots, j$ . If  $j$  rated earlier in the sequence, she might look more informative, as her ratings might be less redundant with the information provided by previous raters.

### 5.2 Analyzing the information loss

In order to simplify the analysis, we assume a fixed rating order  $1, 2, \dots, j, \dots$  on all items. We also assume that the ratings are reported without error and the underlying recommender algorithm correctly processes them to determine  $q$  values. These assumptions allow us to present a concise

<sup>3</sup>It is possible that  $\hat{\pi}_{j-1}$  has two or more components in which the expected value of  $l$  is exactly the same, and hence,  $j - 1$  rates in exactly the same way. However, as  $q_{j-1}$  takes the same value on all these components of  $\hat{\pi}_{j-1}$ , we can think of them as one component without altering the analysis.

information-theoretic bound on the information loss *due to influence limiting*. Note that there may be other sources of information loss in practice (*e.g.*, rating errors or recommender imperfections). As before, we assume that the ratings on different items are independent.

Given the order in which users rate, we want to relate the influence  $j$  earns to the informativeness  $I(\hat{\pi}_j || \hat{\pi}_{j-1})$  of  $j$ 's private information. From step 2.f of the algorithm, the expected growth in reputation of a rater  $j$  with reputation  $R_j \geq 1$ , whose rating identifies a component of  $\hat{\pi}_j$  on which the prediction changes from  $u$  to  $q$ , is:

$$q[L(HI, u) - L(HI, q)] + (1 - q)[L(LO, u) - L(LO, q)]$$

This is just the relative entropy,  $D(q||u)$ .

The players' reputations are initially very small. Initially, then, the amount that a reputation grows is scaled by  $R_j$ . We now show that a reputation builds exponentially in the number of ratings: The expected logarithm of the rater's reputation grows at a rate that depends on the rater's informativeness but not on the reputation.

Suppose a player with reputation  $R_j < 1$  moves a prediction (on some item  $i$ ) from  $u$  to  $q$ . With probability  $q$ , this item will eventually be labelled  $HI$ , and her reputation will be adjusted to  $R'_j = R_j + R_j(L(HI, u) - L(HI, q))$ . Similarly, with probability  $1 - q$ , her reputation will be changed to  $R'_j = R_j + R_j(L(LO, u) - L(LO, q))$ . Then, the expected value of  $\log R'_j$  is given by:

$$\begin{aligned} E(\log R'_j) &= q \log[R_j(1 + L(HI, u) - L(HI, q))] \\ &\quad + (1 - q) \log[R_j(1 + L(LO, u) - L(LO, q))] \\ &= \log R_j + GF(q||u), \end{aligned}$$

where the *growth factor*  $GF(q||u)$  is defined as

$$\begin{aligned} GF(q||u) &\stackrel{\text{def}}{=} q \log(1 + L(HI, u) - L(HI, q)) \\ &\quad + (1 - q) \log(1 + L(LO, u) - L(LO, q)) \end{aligned}$$

The growth factor determines the expected rate of growth of  $j$ 's reputation a single component of  $\hat{\pi}_j$  that  $j$ 's rating identifies. We define an *expected growth factor* measure  $EGF(\hat{\pi}_j || \hat{\pi}_{j-1})$  by averaging over all possible components that  $j$ 's rating could identify:

$$EGF(\hat{\pi}_j || \hat{\pi}_{j-1}) \stackrel{\text{def}}{=} \sum_{s \in \hat{\pi}_j} p_s GF(q_j(s) || q_{j-1}(s))$$

As in the case of the informativeness measure, we can extend this definition naturally to define  $EGF(\mathbf{q}||\mathbf{u})$  for any functions  $\mathbf{q}(\omega)$  and  $\mathbf{u}(\omega)$  that are constant on components of  $\hat{\pi}_j$  and  $\hat{\pi}_{j-1}$  respectively.

Lemma 5 shows that the growth factor on any single component of  $\hat{\pi}_j$  is close to the relative entropy on that component. Lemma 6 extends this to show that the expected growth factor is close to rater  $j$ 's informativeness  $I(\hat{\pi}_j || \hat{\pi}_{j-1})$ , even if  $j$  is scored relative to the previous influence-limited prediction  $\tilde{q}_{j-1}$  instead of the previous accurate prediction  $q_{j-1}$ . Proofs of these results and theorem 7 are omitted from the paper due to space restrictions; they can be found in an Appendix which has been archived online at <http://hdl.handle.net/2027.42/55415>.

**LEMMA 5.** *For the quadratic scoring rule (MSE) loss, for all  $q, u \in [0, 1]$ ,  $GF(q||u) \geq \frac{D(q||u)}{2}$ .*

**LEMMA 6.** *Suppose  $\hat{\pi}_j$  and  $\hat{\pi}_{j-1}$  are two partitions such that  $\hat{\pi}_j$  is a refinement of  $\hat{\pi}_{j-1}$ . For each state  $\omega$ , let  $\mathbf{q}_j(\omega) = E(l(\omega) | \hat{\pi}_j)$  be the optimal prediction function given partition  $\hat{\pi}_j$ . Let  $\mathbf{u}(\omega)$  be any function that is constant on each component of  $\hat{\pi}_{j-1}$ . Then,  $EGF(\mathbf{q}_j || \mathbf{u}) \geq I(\hat{\pi}_j || \hat{\pi}_{j-1})/2$  in the quadratic loss model.*

**THEOREM 7.** *Suppose rater  $j$  has rated  $m$  items, and suppose the informativeness of rater  $j$  is  $I(q_j || q_{j-1}) = h$ . Then, for all  $m \geq (2\lambda + 1)/h$ , rater  $j$ 's expected reputation (with the quadratic scoring rule) is bounded below by*

$$E(R_j) \geq mh - 2\lambda - 2 \log(mh - 2\lambda)$$

The intuition behind this bound is simple: While  $j$ 's reputation is below 1,  $\log R_j$  increases by at least  $h/2$  in expectation; thus, after  $2\lambda/h$  ratings,  $\log R_j$  will be at least 0 (*i.e.*,  $R_j$  will be at least 1) in expectation. After this,  $j$  is not influence limited, and she will earn (an expected amount of)  $h$  in reputation in each subsequent round. Thus, in each of the last  $(m - 2\lambda/h)$  rounds, she gains reputation  $h$  in expectation, for a total of about  $(mh - 2\lambda)$ .

An important consequence of Theorem 7 is that, by the Reward-Performance inequality (Lemma 2), the expected impact of player  $j$  will be greater than her reputation. In other words, we can expect to discard *at most* about  $2\lambda$  units of information from each rater in total.

Note that this information loss bound employs a very conservative accounting scheme. Without the Influence Limiter, rater  $j$  would have contributed  $h$  bits on item  $i$ , but the influence limits reduces the effect of  $j$ 's rating, we account for this as lost bits. However, if there is a subsequent rater  $j+1$  with sufficient reputation, the eventual prediction may be the same as without influence limits. Rater  $j+1$  might have been redundant in the absence of influence limits, but contributes valuable information to compensate for the loss of  $j$ 's information when influence limits are present. Thus, the information loss bounds are based on a worst-case scenario.

## 6. CONCLUSION AND FUTURE WORK

We have presented an architecture and algorithm to limit the influence of individual raters that is provably manipulation-resistant (even when an attacker can create a large but bounded number of sybils), yet discards only a bounded amount of information from each rater. The smaller the size of the sybil attack that must be prevented, the less information that will be discarded from honest raters as they build up their credibility. There are two key ingredients to our system. First, we use the same metric  $L$  to measure the prediction performance of the system and to calculate raters' contributions (reputations). Second, the bootstrapping scheme is carefully balanced: we tie the influence a rater can have to her accumulated reputation in a way that enables informative raters to ramp up exponentially fast, while limiting the total damage at any point to the amount of positive impact already created by each rater.

Our results suggest several questions for future work. We have presented the algorithm as if one item is under consideration at any point of time, or equivalently, as if we get feedback about the true label as soon as the recommendation is used. In practice, several recommendations may be used before we get feedback about any of them; our algo-

rithm will need to be modified slightly to manage “credit” in this scenario.

Another complication that we have not yet addressed is the target’s selection bias in labeling items. The target is more likely to view items with high recommendations, and as a result, items with low recommendation values might be less likely to be labelled; this could influence the raters’ incentives to provide effort. Modified scoring schemes that correct for this bias would be very interesting.

Our manipulation resistance result assumes that an attacker cannot mislead future raters into amplifying the effects of their ratings in a way that boosts the attackers’ own credibility. Another area of interest for future work is to analyze and prevent such non-myopic strategies.

Although we have proven theoretical bounds on information loss, any amount of discarded information during the initiation period is still of concern, particularly in settings in which each individual rater has low incremental informativeness  $h$  or will rate only a few items. The algorithm will need to be carefully tuned to work well in such settings. A lower bound on the minimum information loss that any manipulation-resistant algorithm must incur would also be useful for comparison.

## Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. IIS 0308006.

## 7. REFERENCES

- [1] B. Awerbuch and R. D. Kleinberg. Competitive collaborative learning. In *18th Annual Conference on Learning Theory (COLT 2005)*, volume 3559 of *LNAI*, pages 233–248. Springer, 2005.
- [2] B. Awerbuch, B. Patt-Shamir, D. Peleg, and M. R. Tuttle. Improved recommendation systems. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1174–1183. SIAM, 2005.
- [3] R. Bhattacharjee and A. Goel. Algorithms and incentives for robust ranking. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA ’07)*, 2007.
- [4] G. Brier. Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, 78:1–3, 1950.
- [5] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.
- [6] A. Cheng and E. Friedman. Sybilproof reputation mechanisms. In *P2PECON ’05: Proceeding of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*, pages 128–132, 2005.
- [7] P.-A. Chirita, W. Nejdl, and C. Zamfir. Preventing shilling attacks in online recommender systems. In *WIDM 05*, pages 67–74, 2005.
- [8] C. Dellarocas. Building trust online: The design of robust reputation reporting mechanisms in online trading communities. In *Information Society or Information Economy? A combined perspective on the digital era*, pages 95–113. Idea Book Publishing, 2003.
- [9] I. J. Good. Rational decisions. *Journal of the Royal Statistical Society. Series B (Methodological)*, 14(1):107–114, 1952.
- [10] P. Grunwald and A. Dawid. Game theory, maximum entropy, minimum discrepancy and robust bayesian decision theory. *Annals of Statistics*, 32:1367–1433, 2004.
- [11] R. Hanson. Combinatorial information market design. *Information Systems Frontiers*, 5(1):107–119, 2003.
- [12] R. Hanson, R. Oprea, and D. Porter. Information aggregation and manipulation in an experimental market. *Journal of Economic Behavior and Organization*, page (to appear), 2006.
- [13] J. Herlocker, J. Konstan, and J. Riedl. An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms. *Information Retrieval*, 5:287–310, 2002.
- [14] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in P2P networks. In *Proceedings of WWW ’03*, pages 640–651, 2003.
- [15] S. K. Lam and J. Riedl. Shilling recommender systems for fun and profit. In *Proceedings of WWW ’04*, pages 393–402, 2004.
- [16] R. Levien. *Attack-Resistant Trust Metrics*. PhD thesis, University of California, Berkeley, 2004.
- [17] P. Massa and B. Bhattacharjee. Using trust in recommender systems: An experimental analysis. In *Proceedings of the 2nd International Conference on Trust Management*, 2004.
- [18] B. Mehta, T. Hoffman, and P. Fankhauser. Lies and propaganda: detecting spam users in collaborative filtering. In *Proceedings of IUI’07*, 2007.
- [19] N. Miller, P. Resnick, and R. Zeckhauser. Eliciting honest feedback: The peer-prediction method. *Management Science*, 51(9):1359–1373, 2005.
- [20] B. Mobasher, R. Burke, R. Bhaumik, and C. Williams. Towards trustworthy recommender systems: An analysis of attack models and algorithm robustness. *ACM Transactions on Internet Technology*, 7(2):1–40, 2007.
- [21] J. O’Donovan and B. Smyth. Trust no one: Evaluating trust-based filtering for recommenders. In *Proceedings of IJCAI’05*, 2005.
- [22] M. O’Mahony, N. Hurley, and G. Silvestre. Promoting recommendations: An attack on collaborative filtering. In *Proceedings of the 13th International Conference on Database and Expert System Applications*, pages 494–503. Springer-Verlag, 2002.
- [23] M. P. O’Mahony, N. J. Hurley, and G. C. M. Silvestre. Detecting noise in recommender system databases. In *Proceedings of the 2006 International Conference on Intelligent User Interfaces*, pages 109–115, 2006.
- [24] A. M. Rashid, I. Albert, D. Cosley, S. K. Lam, S. M. McNee, J. A. Konstan, and J. Riedl. Getting to know you: learning new user preferences in recommender systems. In *IUI ’02*, pages 127–134, 2002.
- [25] A. M. Rashid, G. Karypis, and J. Riedl. Influence in ratings-based recommender systems: An algorithm-independent approach. In *Proceedings of the SIAM International Conference on Data Mining*, 2005.
- [26] G. Shafer and V. Vovk. *Probability and Finance: It’s Only a Game!* John Wiley and Sons, 2001.
- [27] L. von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using Hard AI Problems for Security. In *Proceedings of Eurocrypt 2003*, 2003.

# Item-Based Collaborative Filtering Recommendation Algorithms

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl

{sarwar, karypis, konstan, riedl}@cs.umn.edu

GroupLens Research Group/Army HPC Research Center

Department of Computer Science and Engineering

University of Minnesota, Minneapolis, MN 55455

## ABSTRACT

Recommender systems apply knowledge discovery techniques to the problem of making personalized recommendations for information, products or services during a live interaction. These systems, especially the k-nearest neighbor collaborative filtering based ones, are achieving widespread success on the Web. The tremendous growth in the amount of available information and the number of visitors to Web sites in recent years poses some key challenges for recommender systems. These are: producing high quality recommendations, performing many recommendations per second for millions of users and items and achieving high coverage in the face of data sparsity. In traditional collaborative filtering systems the amount of work increases with the number of participants in the system. New recommender system technologies are needed that can quickly produce high quality recommendations, even for very large-scale problems. To address these issues we have explored item-based collaborative filtering techniques. Item-based techniques first analyze the user-item matrix to identify relationships between different items, and then use these relationships to indirectly compute recommendations for users.

In this paper we analyze different item-based recommendation generation algorithms. We look into different techniques for computing item-item similarities (e.g., item-item correlation vs. cosine similarities between item vectors) and different techniques for obtaining recommendations from them (e.g., weighted sum vs. regression model). Finally, we experimentally evaluate our results and compare them to the basic k-nearest neighbor approach. Our experiments suggest that item-based algorithms provide dramatically better performance than user-based algorithms, while at the same time providing better quality than the best available user-based algorithms.

## 1. INTRODUCTION

The amount of information in the world is increasing far more quickly than our ability to process it. All of us have known the feeling of being overwhelmed by the number of new books, journal articles, and conference proceedings coming out each year. Technology has dramatically reduced the barriers to publishing and distributing information. Now it is time to create the technologies that can help us sift

Copyright is held by the author/owner.  
WWW10, May 1-5, 2001, Hong Kong.  
ACM 1-58113-348-0/01/0005.

through all the available information to find that which is most valuable to us.

One of the most promising such technologies is *collaborative filtering* [19, 27, 14, 16]. Collaborative filtering works by building a database of preferences for items by users. A new user, Neo, is matched against the database to discover *neighbors*, which are other users who have historically had similar taste to Neo. Items that the neighbors like are then recommended to Neo, as he will probably also like them. Collaborative filtering has been very successful in both research and practice, and in both information filtering applications and E-commerce applications. However, there remain important research questions in overcoming two fundamental challenges for collaborative filtering recommender systems.

The first challenge is to improve the scalability of the collaborative filtering algorithms. These algorithms are able to search tens of thousands of potential neighbors in real-time, but the demands of modern systems are to search tens of millions of potential neighbors. Further, existing algorithms have performance problems with individual users for whom the site has large amounts of information. For instance, if a site is using browsing patterns as indications of content preference, it may have thousands of data points for its most frequent visitors. These "long user rows" slow down the number of neighbors that can be searched per second, further reducing scalability.

The second challenge is to improve the quality of the recommendations for the users. Users need recommendations they can trust to help them find items they will like. Users will "vote with their feet" by refusing to use recommender systems that are not consistently accurate for them.

In some ways these two challenges are in conflict, since the less time an algorithm spends searching for neighbors, the more scalable it will be, and the worse its quality. For this reason, it is important to treat the two challenges simultaneously so the solutions discovered are both useful and practical.

In this paper, we address these issues of recommender systems by applying a different approach—item-based algorithm. The bottleneck in conventional collaborative filtering algorithms is the search for neighbors among a large user population of potential neighbors [12]. Item-based algorithms avoid this bottleneck by exploring the relationships between items first, rather than the relationships between users. Recommendations for users are computed by finding items that are similar to other items the user has liked. Because the relationships between items are relatively static,

item-based algorithms may be able to provide the same quality as the user-based algorithms with less online computation.

## 1.1 Related Work

In this section we briefly present some of the research literature related to collaborative filtering, recommender systems, data mining and personalization.

Tapestry [10] is one of the earliest implementations of collaborative filtering-based recommender systems. This system relied on the explicit opinions of people from a close-knit community, such as an office workgroup. However, recommender system for large communities cannot depend on each person knowing the others. Later, several ratings-based automated recommender systems were developed. The GroupLens research system [19, 16] provides a pseudonymous collaborative filtering solution for Usenet news and movies. Ringo [27] and Video Recommender [14] are email and web-based systems that generate recommendations on music and movies, respectively. A special issue of Communications of the ACM [20] presents a number of different recommender systems.

Other technologies have also been applied to recommender systems, including Bayesian networks, clustering, and Horting. Bayesian networks create a model based on a training set with a decision tree at each node and edges representing user information. The model can be built off-line over a matter of hours or days. The resulting model is very small, very fast, and essentially as accurate as nearest neighbor methods [6]. Bayesian networks may prove practical for environments in which knowledge of user preferences changes slowly with respect to the time needed to build the model but are not suitable for environments in which user preference models must be updated rapidly or frequently.

Clustering techniques work by identifying groups of users who appear to have similar preferences. Once the clusters are created, predictions for an individual can be made by averaging the opinions of the other users in that cluster. Some clustering techniques represent each user with partial participation in several clusters. The prediction is then an average across the clusters, weighted by degree of participation. Clustering techniques usually produce less-personal recommendations than other methods, and in some cases, the clusters have worse accuracy than nearest neighbor algorithms [6]. Once the clustering is complete, however, performance can be very good, since the size of the group that must be analyzed is much smaller. Clustering techniques can also be applied as a "first step" for shrinking the candidate set in a nearest neighbor algorithm or for distributing nearest-neighbor computation across several recommender engines. While dividing the population into clusters may hurt the accuracy or recommendations to users near the fringes of their assigned cluster, pre-clustering may be a worthwhile trade-off between accuracy and throughput.

Horting is a graph-based technique in which nodes are users, and edges between nodes indicate degree of similarity between two users [1]. Predictions are produced by walking the graph to nearby nodes and combining the opinions of the nearby users. Horting differs from nearest neighbor as the graph may be walked through other users who have not rated the item in question, thus exploring transitive relationships that nearest neighbor algorithms do not consider.

In one study using synthetic data, Horting produced better predictions than a nearest neighbor algorithm [1].

Schafer et al., [26] present a detailed taxonomy and examples of recommender systems used in E-commerce and how they can provide one-to-one personalization and at the same can capture customer loyalty. Although these systems have been successful in the past, their widespread use has exposed some of their limitations such as the problems of sparsity in the data set, problems associated with high dimensionality and so on. Sparsity problem in recommender system has been addressed in [23, 11]. The problems associated with high dimensionality in recommender systems have been discussed in [4], and application of dimensionality reduction techniques to address these issues has been investigated in [24].

Our work explores the extent to which item-based recommenders, a new class of recommender algorithms, are able to solve these problems.

## 1.2 Contributions

This paper has three primary research contributions:

1. Analysis of the item-based prediction algorithms and identification of different ways to implement its sub-tasks.
2. Formulation of a precomputed model of item similarity to increase the online scalability of item-based recommendations.
3. An experimental comparison of the quality of several different item-based algorithms to the classic user-based (nearest neighbor) algorithms.

## 1.3 Organization

The rest of the paper is organized as follows. The next section provides a brief background in collaborative filtering algorithms. We first formally describe the collaborative filtering process and then discuss its two variants memory-based and model-based approaches. We then present some challenges associated with the memory-based approach. In section 3, we present the item-based approach and describe different sub-tasks of the algorithm in detail. Section 4 describes our experimental work. It provides details of our data sets, evaluation metrics, methodology and results of different experiments and discussion of the results. The final section provides some concluding remarks and directions for future research.

## 2. COLLABORATIVE FILTERING BASED RECOMMENDER SYSTEMS

*Recommender systems* apply data analysis techniques to the problem of helping users find the items they would like to purchase at E-Commerce sites by producing a predicted likeliness score or a list of *top-N* recommended items for a given user. Item recommendations can be made using different methods. Recommendations can be based on demographics of the users, overall top selling items, or past buying habit of users as a predictor of future items. Collaborative Filtering (CF) [19, 27] is the most successful recommendation technique to date. The basic idea of CF-based algorithms is to provide item recommendations or predictions based on the opinions of other like-minded

users. The opinions of users can be obtained *explicitly* from the users or by using some *implicit* measures.

### 2.0.1 Overview of the Collaborative Filtering Process

The goal of a collaborative filtering algorithm is to suggest new items or to predict the utility of a certain item for a particular user based on the user's previous likings and the opinions of other like-minded users. In a typical CF scenario, there is a list of  $m$  users  $\mathcal{U} = \{u_1, u_2, \dots, u_m\}$  and a list of  $n$  items  $\mathcal{I} = \{i_1, i_2, \dots, i_n\}$ . Each user  $u_i$  has a list of items  $I_{u_i}$ , which the user has expressed his/her opinions about. Opinions can be explicitly given by the user as a *rating score*, generally within a certain numerical scale, or can be implicitly derived from purchase records, by analyzing timing logs, by mining web hyperlinks and so on [28, 16]. Note that  $I_{u_i} \subseteq \mathcal{I}$  and it is possible for  $I_{u_i}$  to be a *null-set*. There exists a distinguished user  $u_a \in \mathcal{U}$  called the *active user* for whom the task of a collaborative filtering algorithm is to find an item likeliness that can be of two forms.

- **Prediction** is a numerical value,  $P_{a,j}$ , expressing the predicted likeliness of item  $i_j \notin I_{u_a}$  for the active user  $u_a$ . This predicted value is within the same scale (e.g., from 1 to 5) as the opinion values provided by  $u_a$ .
- **Recommendation** is a list of  $N$  items,  $I_r \subset \mathcal{I}$ , that the active user will like the most. Note that the recommended list must be on items not already purchased by the active user, i.e.,  $I_r \cap I_{u_a} = \emptyset$ . This interface of CF algorithms is also known as *Top-N recommendation*.

Figure 1 shows the schematic diagram of the collaborative filtering process. CF algorithms represent the entire  $m \times n$  user-item data as a ratings matrix,  $\mathcal{A}$ . Each entry  $a_{i,j}$  in  $\mathcal{A}$  represents the preference score (ratings) of the  $i$ th user on the  $j$ th item. Each individual ratings is within a numerical scale and it can as well be 0 indicating that the user has not yet rated that item. Researchers have devised a number of collaborative filtering algorithms that can be divided into two main categories—*Memory-based (user-based)* and *Model-based (item-based)* algorithms [6]. In this section we provide a detailed analysis of CF-based recommender system algorithms.

**Memory-based Collaborative Filtering Algorithms.** Memory-based algorithms utilize the entire user-item database to generate a prediction. These systems employ statistical techniques to find a set of users, known as *neighbors*, that have a history of agreeing with the target user (i.e., they either rate different items similarly or they tend to buy similar set of items). Once a neighborhood of users is formed, these systems use different algorithms to combine the preferences of neighbors to produce a prediction or *top-N* recommendation for the active user. The techniques, also known as *nearest-neighbor* or user-based collaborative filtering, are more popular and widely used in practice.

**Model-based Collaborative Filtering Algorithms.** Model-based collaborative filtering algorithms provide item recommendation by first developing a model of user ratings. Algorithms in this category take a probabilistic approach and envision the collaborative filtering process as computing the expected value of a user prediction, given his/her ratings

on other items. The model building process is performed by different *machine learning* algorithms such as **Bayesian network**, **clustering**, and **rule-based** approaches. The Bayesian network model [6] formulates a probabilistic model for collaborative filtering problem. Clustering model treats collaborative filtering as a classification problem [2, 6, 29] and works by clustering similar users in same class and estimating the probability that a particular user is in a particular class  $C$ , and from there computes the conditional probability of ratings. The rule-based approach applies association rule discovery algorithms to find association between co-purchased items and then generates item recommendation based on the strength of the association between items [25].

### 2.0.2 Challenges of User-based Collaborative Filtering Algorithms

User-based collaborative filtering systems have been very successful in past, but their widespread use has revealed some potential challenges such as:

- **Sparsity.** In practice, many commercial recommender systems are used to evaluate large item sets (e.g., Amazon.com recommends books and CDnow.com recommends music albums). In these systems, even active users may have purchased well under 1% of the items (1% of 2 million books is 20,000 books). Accordingly, a recommender system based on nearest neighbor algorithms may be unable to make any item recommendations for a particular user. As a result the accuracy of recommendations may be poor.
- **Scalability.** Nearest neighbor algorithms require computation that grows with both the number of users and the number of items. With millions of users and items, a typical web-based recommender system running existing algorithms will suffer serious scalability problems.

The weakness of nearest neighbor algorithm for large, sparse databases led us to explore alternative recommender system algorithms. Our first approach attempted to bridge the sparsity by incorporating semi-intelligent filtering agents into the system [23, 11]. These agents evaluated and rated each item using syntactic features. By providing a dense ratings set, they helped alleviate coverage and improved quality. The filtering agent solution, however, did not address the fundamental problem of poor relationships among like-minded but sparse-rating users. To explore that we took an algorithmic approach and used Latent Semantic Indexing (LSI) to capture the similarity between users and items in a reduced dimensional space [24, 25]. In this paper we look into another technique, the model-based approach, in addressing these challenges, especially the scalability challenge. The main idea here is to analyze the user-item representation matrix to identify relations between different items and then to use these relations to compute the prediction score for a given user-item pair. The intuition behind this approach is that a user would be interested in purchasing items that are similar to the items the user liked earlier and would tend to avoid items that are similar to the items the user didn't like earlier. These techniques don't require to identify the neighborhood of similar users when a recommendation is requested; as a result they tend to produce much faster recommendations. A number of different

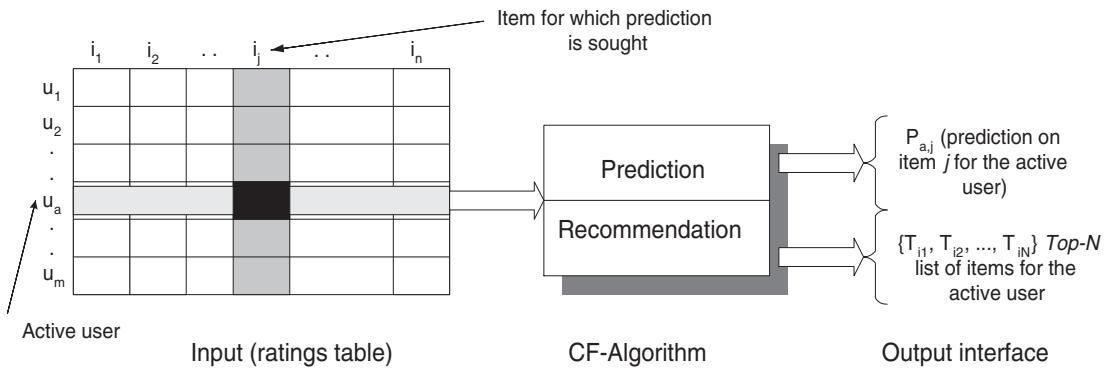


Figure 1: The Collaborative Filtering Process.

schemes have been proposed to compute the association between items ranging from probabilistic approach [6] to more traditional item-item correlations [15, 13]. We present a detailed analysis of our approach in the next section.

### 3. ITEM-BASED COLLABORATIVE FILTERING ALGORITHM

In this section we study a class of item-based recommendation algorithms for producing predictions to users. Unlike the user-based collaborative filtering algorithm discussed in Section 2, the item-based approach looks into the set of items the target user has rated and computes how similar they are to the target item  $i$  and then selects  $k$  most similar items  $\{i_1, i_2, \dots, i_k\}$ . At the same time their corresponding similarities  $\{s_{i1}, s_{i2}, \dots, s_{ik}\}$  are also computed. Once the most similar items are found, the prediction is then computed by taking a weighted average of the target user's ratings on these similar items. We describe these two aspects, namely, the similarity computation and the prediction generation in details here.

#### 3.1 Item Similarity Computation

One critical step in the item-based collaborative filtering algorithm is to compute the similarity between items and then to select the most similar items. The basic idea in similarity computation between two items  $i$  and  $j$  is to first isolate the users who have rated both of these items and then to apply a similarity computation technique to determine the similarity  $s_{ij}$ . Figure 2 illustrates this process; here the matrix rows represent users and the columns represent items.

There are a number of different ways to compute the similarity between items. Here we present three such methods. These are cosine-based similarity, correlation-based similarity and adjusted-cosine similarity.

##### 3.1.1 Cosine-based Similarity

In this case, two items are thought of as two vectors in the  $m$  dimensional user-space. The similarity between them is measured by computing the cosine of the angle between these two vectors. Formally, in the  $m \times n$  ratings matrix in Figure 2, similarity between items  $i$  and  $j$ , denoted by

$sim(i, j)$  is given by

$$sim(i, j) = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\|_2 * \|\vec{j}\|_2}$$

where ‘‘.’’ denotes the dot-product of the two vectors.

##### 3.1.2 Correlation-based Similarity

In this case, similarity between two items  $i$  and  $j$  is measured by computing the Pearson- $r$  correlation  $corr_{i,j}$ . To make the correlation computation accurate we must first isolate the co-rated cases (i.e., cases where the users rated both  $i$  and  $j$ ) as shown in Figure 2. Let the set of users who both rated  $i$  and  $j$  are denoted by  $U$  then the correlation similarity is given by

$$sim(i, j) = \frac{\sum_{u \in U} (R_{u,i} - \bar{R}_i)(R_{u,j} - \bar{R}_j)}{\sqrt{\sum_{u \in U} (R_{u,i} - \bar{R}_i)^2} \sqrt{\sum_{u \in U} (R_{u,j} - \bar{R}_j)^2}}$$

Here  $R_{u,i}$  denotes the rating of user  $u$  on item  $i$ ,  $\bar{R}_i$  is the average rating of the  $i$ -th item.

##### 3.1.3 Adjusted Cosine Similarity

One fundamental difference between the similarity computation in user-based CF and item-based CF is that in case of user-based CF the similarity is computed along the rows of the matrix but in case of the item-based CF the similarity is computed along the columns, i.e., each pair in the co-rated set corresponds to a different user (Figure 2). Computing similarity using basic cosine measure in item-based case has one important drawback—the differences in rating scale between different users are not taken into account. The adjusted cosine similarity offsets this drawback by subtracting the corresponding user average from each co-rated pair. Formally, the similarity between items  $i$  and  $j$  using this scheme is given by

$$sim(i, j) = \frac{\sum_{u \in U} (R_{u,i} - \bar{R}_u)(R_{u,j} - \bar{R}_u)}{\sqrt{\sum_{u \in U} (R_{u,i} - \bar{R}_u)^2} \sqrt{\sum_{u \in U} (R_{u,j} - \bar{R}_u)^2}}$$

Here  $\bar{R}_u$  is the average of the  $u$ -th user's ratings.

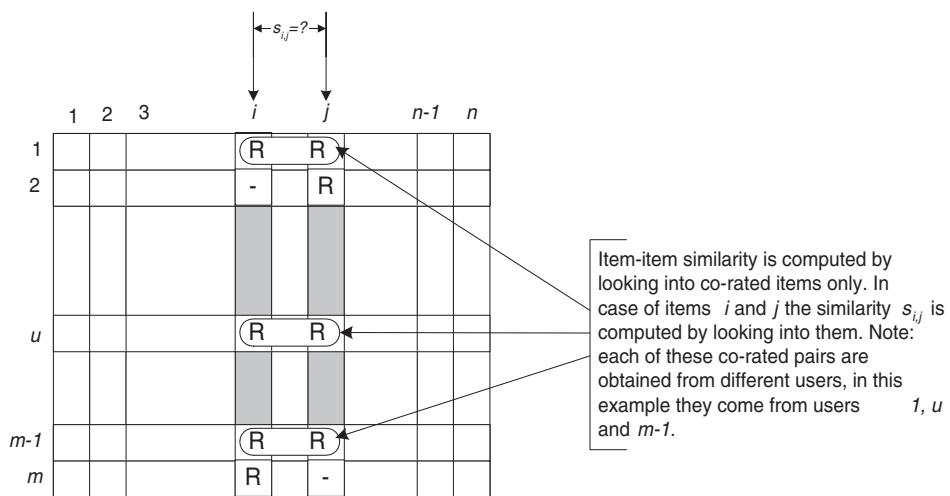


Figure 2: Isolation of the co-rated items and similarity computation

### 3.2 Prediction Computation

The most important step in a collaborative filtering system is to generate the output interface in terms of prediction. Once we isolate the set of most similar items based on the similarity measures, the next step is to look into the target users ratings and use a technique to obtain predictions. Here we consider two such techniques.

#### 3.2.1 Weighted Sum

As the name implies, this method computes the prediction on an item  $i$  for a user  $u$  by computing the sum of the ratings given by the user on the items similar to  $i$ . Each rating is weighted by the corresponding similarity  $s_{i,j}$  between items  $i$  and  $j$ . Formally, using the notion shown in Figure 3 we can denote the prediction  $P_{u,i}$  as

$$P_{u,i} = \frac{\sum_{\text{all similar items, } N} (s_{i,N} * R_{u,N})}{\sum_{\text{all similar items, } N} (|s_{i,N}|)}$$

Basically, this approach tries to capture how the active user rates the similar items. The weighted sum is scaled by the sum of the similarity terms to make sure the prediction is within the predefined range.

#### 3.2.2 Regression

This approach is similar to the weighted sum method but instead of directly using the ratings of similar items it uses an approximation of the ratings based on regression model. In practice, the similarities computed using cosine or correlation measures may be misleading in the sense that two rating vectors may be distant (in Euclidean sense) yet may have very high similarity. In that case using the raw ratings of the “so-called” similar item may result in poor prediction. The basic idea is to use the same formula as the weighted sum technique, but instead of using the similar item  $N$ ’s “raw” ratings values  $R_{u,N}$ ’s, this model uses their approximated values  $R'_{u,N}$  based on a linear regression model. If we denote the respective vectors of the target item  $i$  and the similar item  $N$  by  $R_i$  and  $R_N$  the linear regression model can be expressed as

$$\bar{R}_N = \alpha \bar{R}_i + \beta + \epsilon$$

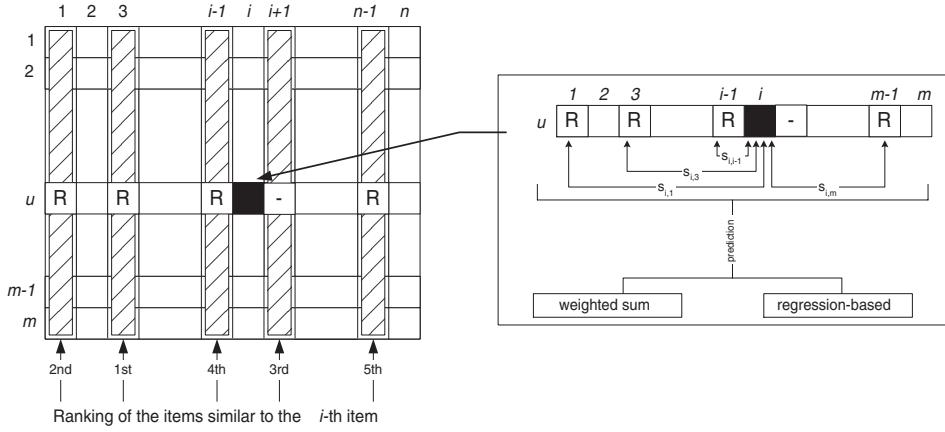
The regression model parameters  $\alpha$  and  $\beta$  are determined by going over both of the rating vectors.  $\epsilon$  is the error of the regression model.

### 3.3 Performance Implications

The largest E-Commerce sites operate at a scale that stresses the direct implementation of collaborative filtering. In neighborhood-based CF systems, the neighborhood formation process, especially the user-user similarity computation step turns out to be the performance bottleneck, which in turn can make the whole process unsuitable for real-time recommendation generation. One way of ensuring high scalability is to use a model-based approach. Model-based systems have the potential to contribute to recommender systems to operate at a high scale. The main idea here to isolate the neighborhood generation and prediction generation steps.

In this paper, we present a model-based approach to precompute item-item similarity scores. The similarity computation scheme is still correlation-based but the computation is performed on the item space. In a typical E-Commerce scenario, we usually have a set of item that is static compared to the number of users that changes most often. The static nature of items leads us to the idea of precomputing the item similarities. One possible way of precomputing the item similarities is to compute all-to-all similarity and then performing a quick table look-up to retrieve the required similarity values. This method, although saves time, requires an  $\mathcal{O}(n^2)$  space for  $n$  items.

The fact that we only need a small fraction of similar items to compute predictions leads us to an alternate model-based scheme. In this scheme, we retain only a small number of similar items. For each item  $j$  we compute the  $k$  most similar items, where  $k \ll n$  and record these item numbers and their similarities with  $j$ . We term  $k$  as the *model size*. Based on this model building step, our prediction generation algorithm works as follows. For generating predictions for a user  $u$  on item  $i$ , our algorithm first retrieves the pre-computed  $k$  most similar items corresponding to the target item  $i$ . Then it looks how many of those  $k$  items were purchased by the user  $u$ , based on this intersection then the



**Figure 3: Item-based collaborative filtering algorithm.** The prediction generation process is illustrated for 5 neighbors

prediction is computed using basic item-based collaborative filtering algorithm.

We observe a quality-performance trade-off here: to ensure good quality we must have a large model size, which leads to the performance problems discussed above. In one extreme, we can have a model size of  $n$ , which will ensure the exact same quality as the original scheme but will have high space complexity. However, our model building step ensures that we retain the most similar items. While generating predictions, these items contribute the most to the prediction scores. Accordingly, we hypothesize that this model-based approach will provide reasonably good prediction quality with even a small model size and hence provide a good performance. We experimentally validate our hypothesis later in this paper. In particular, we experiment with the model size by varying the number of similar items to be stored. Then we perform experiments to compute prediction and response-time to determine the impact of the model size on quality and performance of the whole system.

## 4. EXPERIMENTAL EVALUATION

### 4.1 Data set

We used experimental data from our research website to evaluate different variants of item-based recommendation algorithms.

**Movie data.** We used data from our MovieLens recommender system. MovieLens is a web-based research recommender system that debuted in Fall 1997. Each week hundreds of users visit MovieLens to rate and receive recommendations for movies. The site now has over 43000 users who have expressed opinions on 3500+ different movies. We randomly selected enough users to obtain 100,000 ratings from the database (we only considered users that had rated 20 or more movies). We divided the database into a training set and a test set. For this purpose, we introduced a variable that determines what percentage of data is used as training and test sets; we call this variable  $x$ . A value of  $x = 0.8$  would indicate 80% of the data was used as training set and 20% of the data was used as test set. The data set was converted into a user-item matrix  $A$  that had 943

rows (i.e., 943 users) and 1682 columns (i.e., 1682 movies that were rated by at least one of the users). For our experiments, we also take another factor into consideration, *sparsity level* of data sets. For the data matrix  $R$  This is defined as  $1 - \frac{\text{nonzero entries}}{\text{total entries}}$ . The sparsity level of the Movie data set is, therefore,  $1 - \frac{100,000}{943 \times 1682}$ , which is 0.9369. Throughout the paper we term this data set as  $ML$ .

### 4.2 Evaluation Metrics

Recommender systems research has used several types of measures for evaluating the quality of a recommender system. They can be mainly categorized into two classes:

- *Statistical accuracy metrics* evaluate the accuracy of a system by comparing the numerical recommendation scores against the actual user ratings for the user-item pairs in the test dataset. *Mean Absolute Error* (MAE) between ratings and predictions is a widely used metric. MAE is a measure of the deviation of recommendations from their true user-specified values. For each ratings-prediction pair  $\langle p_i, q_i \rangle$  this metric treats the absolute error between them, i.e.,  $|p_i - q_i|$  equally. The MAE is computed by first summing these absolute errors of the  $N$  corresponding ratings-prediction pairs and then computing the average. Formally,

$$MAE = \frac{\sum_{i=1}^N |p_i - q_i|}{N}$$

The lower the MAE, the more accurately the recommendation engine predicts user ratings. *Root Mean Squared Error* (RMSE), and *Correlation* are also used as statistical accuracy metric.

- *Decision support accuracy metrics* evaluate how effective a prediction engine is at helping a user select high-quality items from the set of all items. These metrics assume the prediction process as a binary operation—either items are predicted (good) or not (bad). With this observation, whether a item has a prediction score of 1.5 or 2.5 on a five-point scale is irrelevant if the user only chooses to consider predictions of 4 or higher. The

most commonly used decision support accuracy metrics are *reversal rate*, *weighted errors* and *ROC sensitivity* [23].

We used MAE as our choice of evaluation metric to report prediction experiments because it is most commonly used and easiest to interpret directly. In our previous experiments [23] we have seen that MAE and ROC provide the same ordering of different experimental schemes in terms of prediction quality.

#### 4.2.1 Experimental Procedure

**Experimental steps.** We started our experiments by first dividing the data set into a training and a test portion. Before starting full experimental evaluation of different algorithms we determined the sensitivity of different parameters to different algorithms and from the sensitivity plots we fixed the optimum values of these parameters and used them for the rest of the experiments. To determine the parameter sensitivity, we work only with the training data and further subdivide it into a training and test portion and carried on our experiments on them. For conducted a 10-fold cross validation of our experiments by randomly choosing different training and test sets each time and taking the average of the MAE values.

**Benchmark user-based system.** To compare the performance of item-based prediction we also entered the training ratings set into a collaborative filtering recommendation engine that employs the Pearson nearest neighbor algorithm (user-user). For this purpose we implemented a flexible prediction engine that implements user-based CF algorithms. We tuned the algorithm to use the best published Pearson nearest neighbor algorithm and configured it to deliver the highest quality prediction without concern for performance (i.e., it considered every possible neighbor to form optimal neighborhoods).

**Experimental platform.** All our experiments were implemented using *C* and compiled using optimization flag  $-O6$ . We ran all our experiments on a linux based PC with Intel Pentium III processor having a speed of 600 MHz and 2GB of RAM.

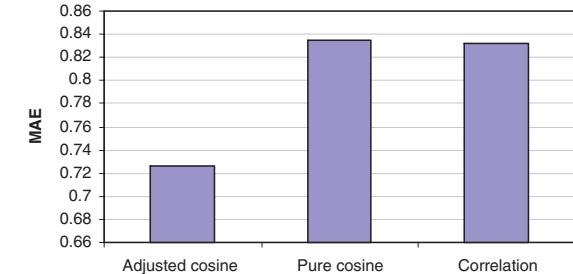
### 4.3 Experimental Results

In this section we present our experimental results of applying item-based collaborative filtering techniques for generating predictions. Our results are mainly divided into two parts—quality results and performance results. In assessing the quality of recommendations, we first determined the sensitivity of some parameters before running the main experiment. These parameters include the neighborhood size, the value of the training/test ratio  $x$ , and effects of different similarity measures. For determining the sensitivity of various parameters, we focused only on the training data set and further divided it into a *training* and a *test* portion and used them to learn the parameters.

#### 4.3.1 Effect of Similarity Algorithms

We implemented three different similarity algorithms basic cosine, adjusted cosine and correlation as described in Section 3.1 and tested them on our data sets. For each simi-

**Relative performance of different similarity measures**



**Figure 4: Impact of the similarity computation measure on item-based collaborative filtering algorithm.**

larity algorithms, we implemented the algorithm to compute the neighborhood and used weighted sum algorithm to generate the prediction. We ran these experiments on our training data and used test set to compute Mean Absolute Error (MAE). Figure 4 shows the experimental results. It can be observed from the results that offsetting the user-average for cosine similarity computation has a clear advantage, as the MAE is significantly lower in this case. Hence, we select the adjusted cosine similarity for the rest of our experiments.

#### 4.3.2 Sensitivity of Training/Test Ratio

To determine the sensitivity of density of the data set, we carried out an experiment where we varied the value of  $x$  from 0.2 to 0.9 in an increment of 0.1. For each of these training/test ratio values we ran our experiments using the two prediction generation techniques—basic weighted sum and regression based approach. Our results are shown in Figure 5. We observe that the quality of prediction increase as we increase  $x$ . The regression-based approach shows better results than the basic scheme for low values of  $x$  but as we increase  $x$  the quality tends to fall below the basic scheme. From the curves, we select  $x = 0.8$  as an optimum value for our subsequent experiments.

#### 4.3.3 Experiments with neighborhood size

The size of the neighborhood has significant impact on the prediction quality [12]. To determine the sensitivity of this parameter, we performed an experiment where we varied the number of neighbors to be used and computed MAE. Our results are shown in Figure 5. We can observe that the size of neighborhood does affect the quality of prediction. But the two methods show different types of sensitivity. The basic item-item algorithm improves as we increase the neighborhood size from 10 to 30, after that the rate of increase diminishes and the curve tends to be flat. On the other hand, the regression-based algorithm shows decrease in prediction quality with increased number of neighbors. Considering both trends we select 30 as our optimal choice of neighborhood size.

#### 4.3.4 Quality Experiments

Once we obtain the optimal values of the parameters, we compare both of our item-based approaches with the benchmark user-based algorithm. We present the results in Figure 6. It can be observed from the charts that the basic

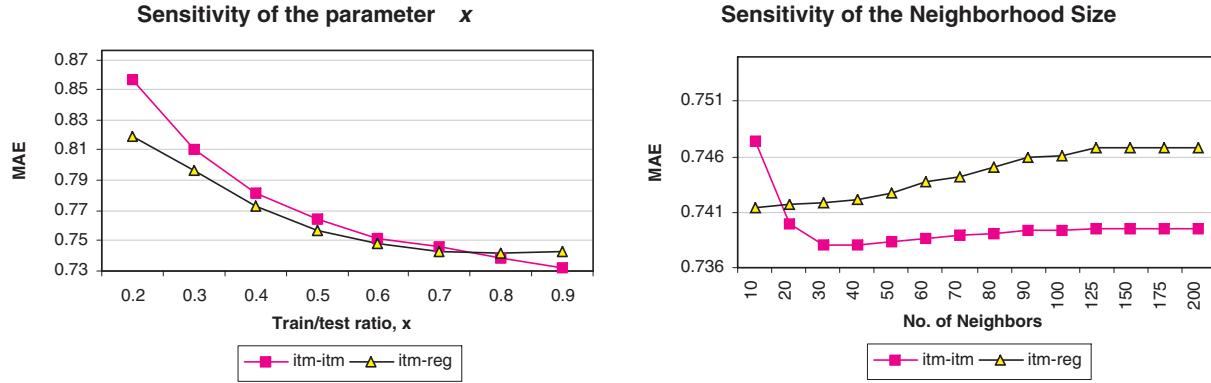


Figure 5: Sensitivity of the parameter  $x$  on the neighborhood size

item-item algorithm out performs the user based algorithm at all values of  $x$  (neighborhood size = 30) and all values of neighborhood size ( $x = 0.8$ ). For example, at  $x = 0.5$  user-user scheme has an MAE of 0.755 and item-item scheme shows an MAE of 0.749. Similarly at a neighborhood size of 60 user-user and item-item schemes show MAE of 0.732 and 0.726 respectively. The regression-based algorithm, however, shows interesting behavior. At low values of  $x$  and at low neighborhood size it out performs the other two algorithms, but as the density of the data set is increased or as we add more neighbors it performs worse, even compared to the user-based algorithm. We also compared our algorithms against the naive nonpersonalized algorithm described in [12].

We draw two conclusions from these results. First, item-based algorithms provide better quality than the user-based algorithms at all sparsity levels. Second, regression-based algorithms perform better with very sparse data set, but as we add more data the quality goes down. We believe this happens as the regression model suffers from data overfitting at high density levels.

#### 4.3.5 Performance Results

After showing that the item-based algorithm provides better quality of prediction than the user-based algorithm, we focus on the scalability issues. As discussed earlier, item-based similarity is more static and allows us to precompute the item neighborhood. This precomputation of the model has certain performance benefits. To make the system even more scalable we looked into the sensitivity of the model size and then looked into the impact of model size on the response time and throughput.

### 4.4 Sensitivity of the Model Size

To experimentally determine the impact of the model size on the quality of the prediction, we selectively varied the number of items to be used for similarity computation from 25 to 200 in an increment of 25. A model size of  $l$  means that we only considered  $l$  best similarity values for model building and later used  $k$  of them for the prediction generation process, where  $k < l$ . Using the training data set we precomputed the item similarity using different model sizes and then used only the weighted sum prediction generation technique to provide the predictions. We then used the test data

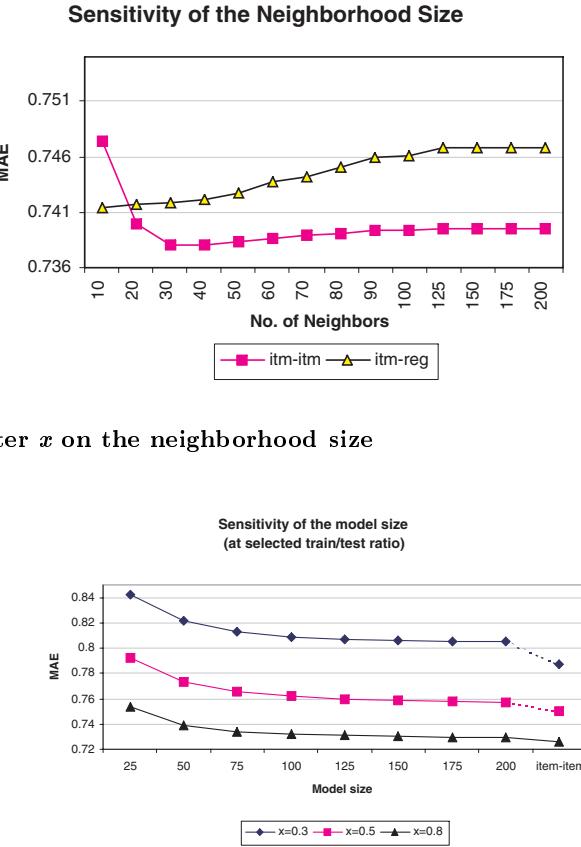


Figure 7: Sensitivity of the model size on item-based collaborative filtering algorithm

set to compute MAE and plotted the values. To compare with the full model size (i.e., model size = no. of items) we also ran the same test considering all similarity values and picked best  $k$  for prediction generation. We repeated the entire process for three different  $x$  values (training/test ratios). Figure 7 shows the plots at different  $x$  values. It can be observed from the plots that the MAE values get better as we increase the model size and the improvements are drastic at the beginning, but gradually slow down as we increase the model size. The most important observation from these plots is the high accuracy that can be achieved using only a fraction of items. For example, at  $x = 0.3$  the full item-item scheme provided an MAE of 0.7873, but using a model size of only 25, we were able to achieve an MAE value of 0.842. At  $x = 0.8$  these numbers are even more appealing—for the full item-item we had an MAE of 0.726 but using a model size of only 25 we were able to obtain an MAE of 0.754, and using a model size of 50 the MAE was 0.738. In other words, at  $x = 0.8$  we were within 96% and 98.3% of the full item-item scheme's accuracy using only 1.9% and 3% of the items, respectively!

This model size sensitivity has important performance implications. It appears from the plots that it is useful to precompute the item similarities using only a fraction of items and yet possible to obtain good prediction quality.

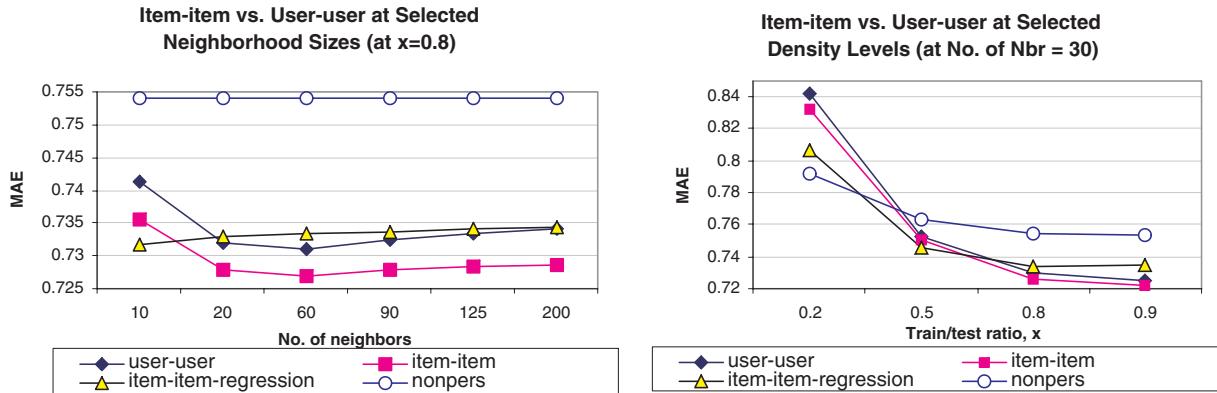


Figure 6: Comparison of prediction quality of *item-item* and *user-user* collaborative filtering algorithms. We compare prediction qualities at  $x = 0.2, 0.5, 0.8$  and  $0.9$ .

#### 4.4.1 Impact of the model size on run-time and throughput

Given the quality of prediction is reasonably good with small model size, we focus on the run-time and throughput of the system. We recorded the time required to generate predictions for the entire test set and plotted them in a chart with varying model size. We plotted the run time at different  $x$  values. Figure 8 shows the plot. Note here that at  $x = 0.25$  the whole system has to make prediction for 25,000 test cases. From the plot we observe a substantial difference in the run-time between the small model size and the full item-item prediction case. For  $x = 0.25$  the run-time is 2.002 seconds for a model size of 200 as opposed to 14.11 for the basic item-item case. This difference is even more prominent with  $x = 0.8$  where a model size of 200 requires only 1.292 seconds and the basic item-item case requires 36.34 seconds.

These run-time numbers may be misleading as we computed them for different training/test ratios where the workload size, i.e., number of predictions to be generated is different (recall that at  $x = 0.3$  our algorithm uses 30,000 ratings as training data and uses the rest of 70,000 ratings as test data to compare predictions generated by the system to the actual ratings). To make the numbers comparable we compute the throughput (predictions generated per second) for the model based and basic item-item schemes. Figure 8 charts these results. We see that for  $x = 0.3$  and at a model size of 100 the system generates 70,000 ratings in 1.487 seconds producing a throughput rate of 47,361 where as the basic item-item scheme produced a throughput of 4961 only. At  $x = 0.8$  these two numbers are 21,505 and 550 respectively.

## 4.5 Discussion

From the experimental evaluation of the item-item collaborative filtering scheme we make some important observations. First, the item-item scheme provides better quality of predictions than the user-user ( $k$ -nearest neighbor) scheme. The improvement in quality is consistent over different neighborhood size and training/test ratio. However, the improvement is not significantly large. The second observation is that the item neighborhood is fairly static, which can be potentially pre-computed, which results in very high on-

line performance. Furthermore, due to the model-based approach, it is possible to retain only a small subset of items and produce reasonably good prediction quality. Our experimental results support that claim. Therefore, the item-item scheme is capable in addressing the two most important challenges of recommender systems for E-Commerce—quality of prediction and high performance.

## 5. CONCLUSION

Recommender systems are a powerful new technology for extracting additional value for a business from its user databases. These systems help users find items they want to buy from a business. Recommender systems benefit users by enabling them to find items they like. Conversely, they help the business by generating more sales. Recommender systems are rapidly becoming a crucial tool in E-commerce on the Web. Recommender systems are being stressed by the huge volume of user data in existing corporate databases, and will be stressed even more by the increasing volume of user data available on the Web. New technologies are needed that can dramatically improve the scalability of recommender systems.

In this paper we presented and experimentally evaluated a new algorithm for CF-based recommender systems. Our results show that item-based techniques hold the promise of allowing CF-based algorithms to scale to large data sets and at the same time produce high-quality recommendations.

## 6. ACKNOWLEDGMENTS

Funding for this research was provided in part by the National Science Foundation under grants IIS 9613960, IIS 9734442, and IIS 9978717 with additional funding by Net Perceptions Inc. This work was also supported by NSF CCR-9972519, EIA-9986042, ACI-9982274 by Army Research Office contract DA/DAAG55-98-1-0441, by the DOE ASCI program and by Army High Performance Computing Research Center contract number DAAH04-95-C-0008. Access to computing facilities was provided by AHPCRC, Minnesota Supercomputer Institute. We like to thank anonymous reviewers for their valuable comments.

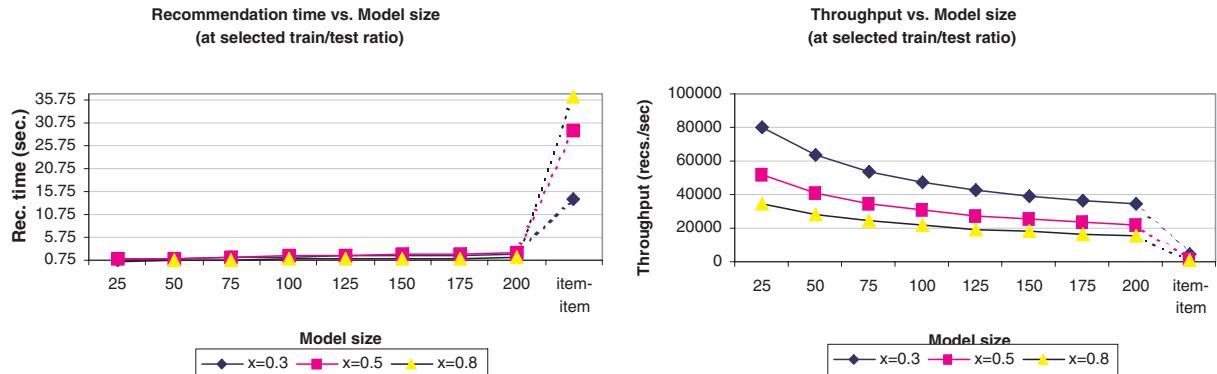


Figure 8: Recommendation time and throughput comparison between model-based scheme and full item-item scheme. The comparisons are shown at three different  $x$  values.

## 7. REFERENCES

- [1] Aggarwal, C. C., Wolf, J. L., Wu, K., and Yu, P. S. (1999). Hortex Hatches an Egg: A New Graph-theoretic Approach to Collaborative Filtering. In *Proceedings of the ACM KDD'99 Conference*. San Diego, CA. pp. 201-212.
- [2] Basu, C., Hirsh, H., and Cohen, W. (1998). Recommendation as Classification: Using Social and Content-based Information in Recommendation. In *Recommender System Workshop'98*. pp. 11-15.
- [3] Berry, M. W., Dumais, S. T., and O'Brian, G. W. (1995). Using Linear Algebra for Intelligent Information Retrieval. *SIAM Review*, 37(4), pp. 573-595.
- [4] Billsus, D., and Pazzani, M. J. (1998). Learning Collaborative Information Filters. In *Proceedings of ICML '98*. pp. 46-53.
- [5] Brachman, R., J., Khabaza, T., Kloesgen, W., Piatetsky-Shapiro, G., and Simoudis, E. 1996. Mining Business Databases. *Communications of the ACM*, 39(11), pp. 42-48, November.
- [6] Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pp. 43-52.
- [7] Cureton, E. E., and D'Agostino, R. B. (1983). Factor Analysis: An Applied Approach. *Lawrence Erlbaum associates publs.* Hillsdale, NJ.
- [8] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41(6), pp. 391-407.
- [9] Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., Eds. (1996). Advances in Knowledge Discovery and Data Mining. *AAAI press/MIT press*.
- [10] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM*. December.
- [11] Good, N., Schafer, B., Konstan, J., Borchers, A., Sarwar, B., Herlocker, J., and Riedl, J. (1999). Combining Collaborative Filtering With Personal Agents for Better Recommendations. In *Proceedings of the AAAI'99 conference*, pp. 439-446.
- [12] Herlocker, J., Konstan, J., Borchers, A., and Riedl, J. (1999). An Algorithmic Framework for Performing Collaborative Filtering. In *Proceedings of ACM SIGIR'99*. ACM press.
- [13] Herlocker, J. (2000). Understanding and Improving Automated Collaborative Filtering Systems. *Ph.D. Thesis, Computer Science Dept., University of Minnesota*.
- [14] Hill, W., Stead, L., Rosenstein, M., and Furnas, G. (1995). Recommending and Evaluating Choices in a Virtual Community of Use. In *Proceedings of CHI '95*.
- [15] Karypis, G. (2000). Evaluation of Item-Based Top-N Recommendation Algorithms. *Technical Report CS-TR-00-46*, Computer Science Dept., University of Minnesota.
- [16] Konstan, J., Miller, B., Maltz, D., Herlocker, J., Gordon, L., and Riedl, J. (1997). GroupLens: Applying Collaborative Filtering to Usenet News. *Communications of the ACM*, 40(3), pp. 77-87.
- [17] Ling, C. X., and Li, C. (1998). Data Mining for Direct Marketing: Problems and Solutions. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pp. 73-79.
- [18] Peppers, D., and Rogers, M. (1997). The One to One Future : Building Relationships One Customer at a Time. *Bantam Doubleday Dell Publishing*.
- [19] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of CSCW '94*, Chapel Hill, NC.
- [20] Resnick, P., and Varian, H. R. (1997). Recommender Systems. Special issue of Communications of the ACM. 40(3).
- [21] Reichheld, F. R., and Sasser Jr., W. (1990). Zero Defections: Quality Comes to Services. *Harvard Business School Review*, 1990(5): pp. 105-111.
- [22] Reichheld, F. R. (1993). Loyalty-Based Management. *Harvard Business School Review*, 1993(2): pp. 64-73.
- [23] Sarwar, B. M., Konstan, J. A., Borchers, A., Herlocker, J., Miller, B., and Riedl, J. (1998). Using

- Filtering Agents to Improve Prediction Quality in the GroupLens Research Collaborative Filtering System. In *Proceedings of CSCW '98*, Seattle, WA.
- [24] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. (2000). Application of Dimensionality Reduction in Recommender System-A Case Study. In *ACM WebKDD 2000 Workshop*.
- [25] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. (2000). Analysis of Recommendation Algorithms for E-Commerce. In *Proceedings of the ACM EC'00 Conference*. Minneapolis, MN. pp. 158-167
- [26] Schafer, J. B., Konstan, J., and Riedl, J. (1999). Recommender Systems in E-Commerce. In *Proceedings of ACM E-Commerce 1999 conference*.
- [27] Shardanand, U., and Maes, P. (1995). Social Information Filtering: Algorithms for Automating 'Word of Mouth'. In *Proceedings of CHI '95*. Denver, CO.
- [28] Terveen, L., Hill, W., Amento, B., McDonald, D., and Creter, J. (1997). PHOAKS: A System for Sharing Recommendations. *Communications of the ACM*, 40(3). pp. 59-62.
- [29] Ungar, L. H., and Foster, D. P. (1998) Clustering Methods for Collaborative Filtering. In *Workshop on Recommender Systems at the 15th National Conference on Artificial Intelligence*.

# The BellKor solution to the Netflix Prize

**Robert M. Bell, Yehuda Koren and Chris Volinsky**  
**AT&T Labs – Research**  
**BellKor@research.att.com**

Our final solution (RMSE=0.8712) consists of blending 107 individual results. Since many of these results are close variants, we first describe the main approaches behind them. Then, we will move to describing each individual result.

The core components of the solution are published in our ICDM'2007 paper [1] (or, KDD-Cup'2007 paper [2]), and also in the earlier KDD'2007 paper [3]. We assume that the reader is familiar with these works and our terminology there.

## Neighborhood-based model (k-NN)

A movie-oriented k-NN approach was thoroughly described in our KDD-Cup'2007 paper [kNN]. We apply it as a post-processor for most other models. Interestingly, it was most effective when applied on residuals of RBMs [5], thereby driving the Quiz RMSE from 0.9093 to 0.8888.

An earlier k-NN approach was described in the KDD'2007 paper ([3], Sec. 3) [Slow-kNN]. It appears that this earlier approach can achieve slightly more accurate results than the newer one, at the expense of a significant increase in running time. Consequently, we dropped the older approach, though some results involving it survive within the final blend.

We also tried more naïve k-NN models, where interpolation weights are based on pairwise similarities between movies (see [2], Sec. 2.2). Specifically, we based weights on  $\text{corr}^2/(1-\text{corr}^2)$  [Corr-kNN], or on  $\text{mse}^{-10}$  [MSE-kNN]. Here,  $\text{corr}$  is the Pearson correlation coefficient between the two respective movies, and  $\text{mse}$  is the mean squared distance between two movies (see definition of  $s_{ij}$  in Sec. 4.1 of [2]). We also tried taking the interpolation weights as the "support-based similarities", which will be defined shortly [Supp-kNN].

Other variants that we tried for computing the interpolation coefficients are: (1) using our KDD-Cup'2007 [2] method on a binary user-movie matrix, which replaces every rating with "1", and sets non-rated user-movie pairs to "0" [Bin-kNN]. (2) Taking results of factorization, and regressing the factors associated with the target movie on the factors associated with its neighbors. Then, the resulting regression coefficients are used as interpolation weights [Fctr-kNN].

As explained in our papers, we also tried user-oriented k-NN approaches. Either in a profound way (see: [1], Sec. 4.3; [3], Sec. 5) [User-kNN], or by just taking weights as pairwise similarities among users [User-MSE-kNN], which is the user-oriented parallel of the aforementioned [MSE-kNN].

Prior to computing interpolation weights, one has to choose the set of neighbors. We find the most similar neighbors based on an appropriate similarity measure. In Sec. 4.1 of [2] we mention a correlation-based similarity measure and a distance-based similarity measure. Another kind of similarity is based solely on who-rated-what (we refer to this as "support-based" or "binary-based" similarity). The full details are given in an Appendix 1. Interestingly, when post-processing residuals of factorization or

RBM<sub>s</sub>, these seemingly inferior support-based similarities led to more accurate results.

## A factorization model

The earlier factorization model that we employed is fully described in our KDD'2007 paper ([3], Sec. 4). Later we moved to a more powerful model, which is described at Section 5.1 of [1]. The essence of these models is alternating between computing all movie factors and all user factors, by optimally solving regularized least squares problems; also known as *alternating least squares*. The KDD'2007 paper [3] suggested computing each factor separately while performing simple shrinkage [IncFctr]. The ICDM'2007 paper [1] suggested computing all factors associated with a single user/movie simultaneously, while using Ridge regression [SimuFctr]. An additional accuracy boost was achieved when we required all factors to be non-negative [NNMF], by employing a non-negative least squares solver. Another alternative that we tried is replacing the Ridge regression (L2-norm penalty) by Lasso regression (L1-norm penalty) [LassoNNMF]. In general, the Lasso regularization was far less effective, but somewhat contributed to our overall blend.

All factorization models significantly benefit from recomputing user-factors in a neighborhood-aware fashion, in the spirit described in our KDD'2007 paper ([3], Sec. 4.4), modified according to the actually employed model. In this process, we adapt the computed user factor to the given query, by weighting all related ratings by movie-movie similarities. Here, choosing the most effective movie-movie similarities involves more art than science. Our recommended choice is taking  $s_{ij} = \text{MSE}(i,j)^{-6}$ , where  $\text{MSE}(i,j)$  is the mean squared distance between ratings of movie i and those of movie j [MseSim]. Other approaches for the movie-movie similarities that were used are: (1)  $s_{ij} = \text{corr}(i,j)^2 / (1 - \text{corr}(i,j)^2)$ , where  $\text{corr}(i,j)$  is the Pearson correlation coefficient [CorrSim]. (2) Compute all similarities simultaneously as we compute interpolation weights in the KDD-Cup'2007 paper ([2], Eq. (13)) [SimuSim]. (3) Basing similarities on the inverse of the edit distance of movie titles, accounting also for gap of release year [EditSim]; see Appendix 2. (4) Support-based similarities, which were mentioned earlier [SuppSim]; see Appendix 1.

In addition, no matter how similarities are computed, we also introduce a "date factor" into them. That is, when measuring the similarity between two ratings, we also account for the date gap between them. More specifically, if movie i was rated  $x$  days later than movie j, we multiply their similarity ( $s_{ij}$ ) by  $\exp(-x/600)$ . The denominator 600 (days) was determined by cross validation, and reflects the fact that after two years, similarity decays by approximately a factor of 3.

### Regularization based on a Gaussian prior:

While Ridge-regression was proved very effective when deriving factors, it offers a quite limited model by assuming that all factors belong to a normal distribution with zero mean, and a uniform diagonal covariance matrix. A richer model, will not impose these restrictions, but assume a general normal distribution for the factors [GaussFctr]; see, e.g., Zhang and J. Koren [6]. In practice, we have found a rich Gaussian prior being very effective when computing user factors, but less so when dealing with movie factors. Consequently, we mainly apply it on the users' side.

## Restricted Boltzmann Machines

Restricted Boltzmann Machines (RBM) for collaborative filtering were recently described by one of the leading teams in this challenge [5]. We implemented their idea, and could verify most claims of the authors. (We used almost the same parameter setting as suggested in [5], except doubling the learning rate to 0.02.) We have found that RBMs lead to competitive results in terms of accuracy, with a relatively low sensitivity to parameter setting. While the basic approach is fully detailed in the paper, one modification that we tried is replacing the multinomial visible units with Gaussian ones. This way the RBMs can post-process the residuals of global effects or any other method.

## Asymmetric factor models

The factorization model offered a symmetric view of users and movies, by directly parameterizing each of them. However, this practice involves a clear redundancy, as user parameters ("factors") are dependent on the movie-parameters, and vice-versa. An interesting family of models differs from the factorization model by parameterizing only the movies, which have higher support in the training data. Consequently, no explicit user factors are computed, but a user factor is implicitly derived by aggregating the movie factors associated with the movies liked by that user (taking the view that a user is "a bag of movies").

This aggregate is often a plain sum of the respective movie factors, transformed by a function that accounts for the deviations in the number of summed values. The approach was first publicly mentioned by Paterek (Section 3.2 of [4], [NSVD1,NSVD2]). Paterek suggested normalizing this sum by the square root of the support of the respective user. Initially, we followed this approach. Later, we found two more efficient related models that we outline in the followings.

### A weighted scores model:

Let  $n$  be the number of movies,  $m$  the number of users, and  $k$  the number of factors. Let us denote by  $g$  the number of different scores ( $g=5$  for the Netflix data).

We estimate  $r_{ui}$ , rating by user  $u$  of movie  $i$ , as follows:

$$r_{ui} = \sum_{f=1}^k \left( p_{if} \cdot \sigma \left( b_f + \sum_{j \text{ rated by } u} w_{score(u,j)} \cdot q_{jf} \right) \right)$$

Here, the constant  $score(u,j)$  is the score given by user  $u$  to movie  $j$  (an integer between 1 to  $g$ ). Whenever this score is unknown, but we know that the rating exists (e.g.,  $(u,j)$  belongs to the Qualifying set), we set  $score(u,j)=0$ .

$\sigma(x)$  is the Sigmoid function, defined as:  $\sigma(x) = \frac{1}{1+e^{-x}}$ .

The following parameters should be learnt from the data:

- (1) Two sets of "movie factors": an  $nxk$  matrix  $P=\{p_{if}\}$ , and an  $nxk$  matrix  $Q=\{q_{if}\}$ .
- (2) A set of  $k$  intercepts - the  $b_f$ 's.
- (3) The  $g+1$  weights:  $w_0, w_1, \dots, w_g$ .

All these  $2kn+k+g+1$  parameters are learnt by gradient descent that minimizes the related cost function:

$$\sum_{\text{given rating } r_{ui}} \left( r_{ui} - \sum_{f=1}^k \left( p_{if} \cdot \sigma \left( b_f + \sum_{j \text{ rated by } u} w_{score(u,j)} \cdot q_{jf} \right) \right) \right)^2$$

Optimization process should be regularized by penalizing the magnitudes of the learnt parameters [SIGMOID1] (learning rate=0.002, regularization=0.01).

One can use a single set of movie factors, by equating the matrices P and Q [SIGMOID2] (learning rate=0.001, regularization=0.01).

Typically, we apply this method to residuals of global effects, so the above  $r_{ui}$  would symbolize a residual, rather than a raw score.

### **Weighting with residuals:**

Let us denote by  $res_{ui}$  the residual of the rating by user u of movie i after removing global effects (one can alternatively just use double centering, which will lead to somewhat inferior results).

We estimate  $res_{ui}$ , the residual of the rating by user u of movie i, as follows:

$$res_{ui} = \sum_{f=1}^k \left( p_{if} \cdot \sigma \left( b_f + \sum_{j \text{ rated by } u} q_{jf} + \sum_{\text{known rating } r_{uj}} res_{uj} \cdot p_{jf} \right) \right)$$

The following parameters should be learnt from the data:

- (1) Two sets of "movie factors": an  $n \times k$  matrix  $P = \{p_{if}\}$ , and an  $n \times k$  matrix  $Q = \{q_{if}\}$ .
- (2) A set of k intercepts - the  $b_f$ 's.

All these  $2kn+k$  parameters are learnt by gradient descent that minimizes the related quadratic error cost function. Optimization process should be regularized by penalizing the magnitudes of the learnt parameters [SIGMOID3] (learning rate=0.001, regularization=0.02).

The astute reader will find distinct relations between this residual-based model and RBMs with Gaussian visible units.

## **Regression models**

Regression can be performed in two symmetric ways:

1. A user-centric approach: The sample is all movies rated by a specific user. The response variable is rating of a movie by this user. The predictor variables are different attributes associated with the movies.
2. A movie-centric approach: The sample is all users that rated a specific movie. The response variable is rating of the movie by a user. The predictor variables are different attributes associated with the users.

The central issue is how to create predictor variables. A few approaches were beneficial to us. The relatively low number of movies facilitates building movie-based predictor variables by concentrating on movie-movie relationships, including:

1. Estimating the movie-movie covariance matrix, and then taking its top k eigenvectors as the predictor variables. This is akin to Principal Components Analysis [PCA].
2. Embedding all movies in a k-D Euclidean space, by solving a multidimensional scaling (MDS) problem, based on minimizing the stress energy by majorization. Typical values of k lie between 40 and 100 [STRESS].
3. Build a user-movie binary matrix, where an entry value is "1" iff the corresponding user rated the corresponding movie. Interestingly, this matrix contains no missing value, so we can directly extract its SVD vectors and use them as predictor variables [BIN-SVD]. In some variants we considered all ratings smaller than 3 as zeros [BIN-SVD3]. When working with these binary-based factors, we suggest normalizing the vector of factors associated with each movie, to offset for movie popularity.

These movie-based predictors were used within a user-centric regression approach. In order to use a movie-centric regression, we need to derive user-based predictors, which are more challenging due to the huge number of users in the dataset. To overcome this, we first derive the movie-based predictors as above, and then infer from them the user-based predictors in one of the following two methods:

1. Regress user-based predictors from the movie-based predictors using a user-centric regression (within the binary representation). This way, [BIN-SVD] and [BIN-SVD3] become [BIN-SVD-USER] and [BIN-SVD3-USER], respectively. Once again, we suggest normalizing the vectors of factors associated with each user.
2. A barycentric assignment: A user predictor is taken as a weighted average of the movie predictors, for movies rated by this user. Here, we used those weights found in the "weighted scores model" mentioned above. This way, [PCA] and [STRESS] become [PCA-USER] and [STRESS-USER], respectively.

Unless stated otherwise, we apply the regression models on residuals of global effects.

## Combining multiple results

Predictive accuracy is substantially improved when blending multiple predictors. *Our experience is that most efforts should be concentrated in deriving substantially different approaches, rather than refining a single technique.* Consequently, our solution is an ensemble of many methods.

We approach blending as a linear regression problem. We ought to regress a target ratings vector on multiple predictors. The target rating vector can be the true ratings of the Probe set, and the predictors are the respective estimates for the Probe set by an ensemble of methods. The solution is the coefficients, or the weights, that should be given to each of the predictors in the ensemble.

An extension would be to slice and dice the target vector based on some criteria. This allows overweighting certain methods on some user-movie pairs, while overweighting different methods on other user-movie pairs. The criterion we used is the support of the user-movie pair, which we define as the minimum between the support associated with the user and the support associated with the movie. (A support of a user refers to the number of ratings made by this user. Similarly, a support of a movie is the number of ratings given to it.) Consequently, we typically partition the Probe (or Qualifying) sets into 15 bins, based on the support level. Then, we solve a different regression

problem within each bin, thereby obtaining unique combination coefficients for each bin. For example, we have found that RBMs should be overweighted on low support pairs, while factorization should be overweighted on high support pairs.

Since the ratings of the Qualifying pairs are unknown, we cannot simply regress them. One can "borrow" the combination coefficients learnt for the Probe set. Accuracy can be improved by partitioning the Probe set into several disjoint sets. Then, learn combination coefficients for each portion of the Probe set separately, while including the rest of the Probe set within the training data. This better reflects the situation with the Qualifying set, where all Probe set is included within the training data. Finally, for the Qualifying coefficients, one can average the coefficients derived for the multiple portions of the Probe set. Alternatively, sufficient statistics that enable regressing directly the Quiz set can be obtained with the aid of the RMSEs reported for each predictor.

## Our predictors

Below, we list all 107 results that were blended to deliver RMSE=0.8712, with their weights within the ensemble. The results are grouped based on the type of method that produced them.

We strongly believe that the success of an ensemble approach depends on the ability of its various predictors to expose different, complementing aspects of the data. Experience shows that this is very different from optimizing the accuracy of each individual predictor. Quite frequently we have found that the more accurate predictors are less useful within the full blend. Therefore, the RMSEs listed below should be considered just as a reference of our experience, and we would not encourage the readers to repeat the same RMSEs, but rather to concentrate on achieving the "spirit" of the listed methods.

Please note that before submitting a result, we translate it so its mean will coincide with the mean of the Quiz set, which is known to be 3.6744<sup>1</sup>. Also, before mixing the results, each of them is centered (to have zero mean). Later, the final blend is translated back so its mean coincides with the Quiz mean.

### Asymmetric factor models

1. *rmse*=0.9194, *weight*=-0.0382  
SIGMOID3 with k=30
2. *rmse*=0.9286, *weight*=-0.0481  
SIGMOID2 with k=40
3. *rmse*=0.9383, *weight*=0.0514  
NSVD2 with k=40
4. *rmse*=0.9245, *weight*=-0.0393  
SIGMOID1 with k=40
5. *rmse*=0.9114, *weight*=0.0574  
40 neighbors kNN on residuals of NSVD1 with k=200.
6. *rmse*=0.9236., *weight*=0.0550  
NSVD1 with k=200
7. *rmse*=0.9259, *weight*=0.0832  
NSVD1 with k=150

---

<sup>1</sup> See Winsteps post at <http://www.netflixprize.com//community/viewtopic.php?id=503>

8.  $rmse=0.9134$ ,  $weight=0.0660$   
30 neighbors kNN on residuals of non-regularized NSVD1 with k=200
9.  $rmse=0.9260$ ,  $weight=-0.0484$   
NSVD1 with k=40

## Regression models

10.  $rmse=0.9269$ ,  $weight=-0.0370$   
PCA-USER, based on top 40 PCs
11.  $rmse=0.9302$ ,  $weight=-0.0499$   
STRESS with 40 coordinates per movie
12.  $rmse=0.9335$ ,  $weight=-0.1145$   
BIN-SVD-USER based on 256 vectors
13.  $rmse=0.8996$ ,  $weight=0.1350$   
50 neighbors kNN on residuals of BIN-SVD-USER (256 vectors)
14.  $rmse=0.9290$ ,  $weight=-0.0626$   
BIN-SVD3-USER based on 65 vectors
15.  $rmse=0.9394$ ,  $weight=0.0311$   
BIN-SVD3-USER based on 256 vectors
16.  $rmse=0.9241$ ,  $weight=-0.0692$   
PCA based on top 40 PCs
17.  $rmse=0.9212$ ,  $weight=-0.0639$   
PCA based on top 50 PCs
18.  $rmse=0.9451$ ,  $weight=-0.0484$   
BIN-SVD-USER based on 60 vectors
19.  $rmse=0.9610$ ,  $weight=0.0401$   
BIN-SVD-USER based on 100 vectors, but here we regressed residuals of double centering rather than the usual residuals of global effects
20.  $rmse=0.9414$ ,  $weight=0.0424$   
BIN-SVD3-USER based on 40 vectors
21.  $rmse=0.9067$ ,  $weight=0.0689$   
20 neighbors Corr-kNN on residuals of BIN-SVD-USER (60 vectors)
22.  $rmse=0.9020$ ,  $weight=0.0556$   
50 neighbors kNN on residuals of BIN-SVD-USER (60 vectors)
23.  $rmse=0.9030$ ,  $weight=-0.0491$   
50 neighbors kNN on residuals of BIN-SVD-USER (100 vectors)
24.  $rmse=0.9223$ ,  $weight=0.0763$   
BIN-SVD3 based on 40 vectors

## Restricted Boltzmann Machines with Gaussian visible units

The default is to apply the machine in a "conditional RBM" mode on data normalized by applying all global effects that we describe in the KDD-Cup'2007 paper ([2], Sec. 3), except the last four, which interact movies/users with popularity.

25.  $rmse=0.9052$ ,  $weight=0.0704$   
800 hidden units
26.  $rmse=0.9044$ ,  $weight=0.0739$   
400 hidden units
27.  $rmse=0.9056$ ,  $weight=0.0771$   
256 hidden units

28.  $rmse=0.9068$ ,  $weight=0.0433$   
100 hidden units
29.  $rmse=0.9121$ ,  $weight=-0.0268$   
40 hidden units
30.  $rmse=0.9429$ ,  $weight=0.0220$   
100 hidden units, applied on raw data (no normalization/centering)
31.  $rmse=0.9489$ ,  $weight=-0.0295$   
50 hidden units, applied on raw data (no normalization/centering)
32.  $rmse=0.9267$ ,  $weight=-0.0726$   
100 hidden units, without conditional RBM, on residuals of full global effects

### Restricted Boltzmann Machines

We use conditional RBMs as described in [5].

33.  $rmse=0.9029$ ,  $weight=0.0785$   
256-unit RBM
34.  $rmse=0.9029$ ,  $weight=-0.0700$   
200-unit RBM
35.  $rmse=0.9093$ ,  $weight=-0.0977$   
100-unit RBM
36.  $rmse=0.9206$ ,  $weight=-0.0239$   
40-unit RBM
- Using RBM as a pre-processor:
37.  $rmse=0.8960$ ,  $weight=0.0577$   
Postprocessing residuals of 100-unit RBM with factorization
38.  $rmse=0.8905$ ,  $weight=0.1839$   
50 neighbors kNN on 200-unit RBM
39.  $rmse=0.8904$ ,  $weight=0.0576$   
40 neighbors kNN on 150-unit RBM
40.  $rmse=0.8888$ ,  $weight=0.1387$   
50 neighbors kNN on 100-unit RBM

### Matrix factorization

41.  $rmse=0.9135$ ,  $weight=-0.0302$   
IncFctr (40 factors)
42.  $rmse=0.8992$ ,  $weight=0.0436$   
IncFctr (80 factors), adaptive user factors by [MseSim]
43.  $rmse=0.9042$ ,  $weight=0.0339$   
IncFctr (40 factors), adaptive user factors by [SuppSim]
44.  $rmse=0.9083$ ,  $weight=-0.0389$   
IncFctr (40 factors), adaptive user by [EditSim]
45.  $rmse=0.9002$ ,  $weight=-0.0907$   
SimuFctr (40 factors), adaptive user factors by [MseSim]
46.  $rmse=0.9050$ ,  $weight=0.1178$   
SimuFctr (40 factors), adaptive user factors with  $s_{ij} = \text{MSE}(i,j)^{-12}$
47.  $rmse=0.9035$ ,  $weight=-0.0546$   
MSE-kNN applied to residuals of SimuFctr (60 factors)
48.  $rmse=0.9515$ ,  $weight=-0.0146$   
NNMF (5 factors)
49.  $rmse=0.9347$ ,  $weight=-0.0227$   
NNMF (20 factors), training set excluded Probe set

50.  $rmse=0.9084$ ,  $weight=-0.0342$   
     NNMF (20 factors), adaptive user factors by [SimuSim]
51.  $rmse=0.9073$ ,  $weight=-0.0353$   
     NNMF (20 factors), adaptive user factors by [EditSim]
52.  $rmse=0.9094$ ,  $weight=-0.1412$   
     NNMF (40 factors)
53.  $rmse=0.9018$ ,  $weight=0.2535$   
     NNMF (40 factors, adaptive user factors by [EditSim])
54.  $rmse=0.8986$ ,  $weight=-0.1093$   
     NNMF (40 factors, adaptive user factors by [MseSim])
55.  $rmse=0.9026$ ,  $weight=-0.1159$   
     Cor-kNN on residuals of NNMF (60 factors)
56.  $rmse=0.8963$ ,  $weight=-0.1032$   
     NNMF (90 factors), adaptive user factors by [MseSim]
57.  $rmse=0.8986$ ,  $weight=0.0714$   
     NNMF (90 factors), adaptive user factors by naive [SuppSim] (where  $x_{ij}=n_i \cdot n_j / n$ )
58.  $rmse=0.9807$ ,  $weight=0.0228$   
     NNMF (90 factors), adaptive user factors by  $s_{ij} = MSE(i,j)^{-1/2}$
59.  $rmse=0.8970$ ,  $weight=0.1028$   
     NNMF (90 factors), adaptive user factors by [SuppSim]
60.  $rmse=0.8978$ ,  $weight=0.1035$   
     NNMF (90 factors), adaptive user factors by [CorrSim]
61.  $rmse=0.8985$ ,  $weight=-0.1121$   
     NNMF (90 factors), adaptive user factors by [EditSim]
62.  $rmse=1.1561$ ,  $weight=-0.0111$   
     NNMF (128 factors), adaptive user factors by [MseSim], but adaptive user factors where computed with Lasso regularization, rather than Ridge regularization
63.  $rmse=0.9039$ ,  $weight=-0.0928$   
     NNMF (128 factors)
64.  $rmse=0.8955$ ,  $weight=0.1060$   
     NNMF (128 factors), adaptive user factors by [MseSim]
65.  $rmse=0.9426$ ,  $weight=-0.0564$   
     LassoNNMF (30 factors)
66.  $rmse=0.9327$ ,  $weight=0.0445$   
     LassoNNMF (30 factors), adaptive user factors by [SuppSim]
67.  $rmse=0.9016$ ,  $weight=0.0543$   
     Start with NNMF 40 factors, then alternate between GaussFctr on user side and SimuFctr on movie side
68.  $rmse=0.8998$ ,  $weight=0.0958$   
     Start with SimuFctr 60 factors, then a single GaussFctr iterations on movie side followed by many GaussFctr iterations on user side
69.  $rmse=0.9070$ ,  $weight=-0.0954$   
     Start with NNMF 90 factors, followed by many GaussFctr iterations on user side
70.  $rmse=0.9098$ ,  $weight=0.0720$   
     Start with SimuFctr 40 factors, followed by many GaussFctr iterations on user side

## Neighborhood-based model (k-NN)

Some k-NN results were embedded in the previous sections. Here, we report the rest.

71.  $rmse=0.8953$ ,  $weight=0.0932$   
30 neighbors kNN on residuals of NNMF (180 factors)
72.  $rmse=0.9105$ ,  $weight=-0.1386$   
50 neighbors kNN on residuals of all global effects except the last 4
73.  $rmse=0.9082$ ,  $weight=-0.0456$   
50 neighbors kNN on residuals of full global effects
74.  $rmse=0.9496$ ,  $weight=-0.0272$   
25 neighbors kNN on raw scores (no normalization)
75.  $rmse=0.8979$ ,  $weight=-0.1099$   
60 neighbors kNN on residuals of NNMF (60 factors)
76.  $rmse=0.9247$ ,  $weight=0.0525$   
50 neighbors Bin-kNN on residuals of full global effects, neighbor selection by [SuppSim]
77.  $rmse=0.9215$ ,  $weight=0.0783$   
50 neighbors Bin-kNN on residuals of full global effects, neighbor selection by [CorrSim]
78.  $rmse=0.9309$ ,  $weight=-0.0985$   
50 neighbors Fctr-kNN on residuals of full global effects. Weights based on 10 factors computed on binary matrix
79.  $rmse=0.9097$ ,  $weight=0.0681$   
25 neighbors Fctr-kNN on residuals of NNMF (60 factors). Weights based on 10 NNMF factors
80.  $rmse=0.9290$ ,  $weight=-0.0451$   
50 neighbors Fctr-kNN on raw scores. Weights based on 10 factors computed on binary matrix
81.  $rmse=0.9097$ ,  $weight=0.0408$   
100 neighbors User-kNN on residuals of NNMF (60 factors)
82.  $rmse=0.9248$ ,  $weight=0.0333$   
30 neighbors User-MSE-kNN on residuals of full global effects
83.  $rmse=0.9057$ ,  $weight=0.0550$   
50 neighbors Slow-kNN on residuals of full global effects
84.  $rmse=0.9170$ ,  $weight=-0.0648$   
Corr-kNN on residuals of full global effects
85.  $rmse=0.9237$ ,  $weight=-0.0561$   
MSE-kNN on residuals of full global effects
86.  $rmse=0.9110$ ,  $weight=0.0439$   
Supp-kNN on residuals of IncFctr (80 factors)
87.  $rmse=0.9440$ ,  $weight=-0.0422$   
Supp-kNN on residuals of full global effects. Here, we used the more naïve similarities where  $x_{ij}=n_i \cdot n_j / n$
88.  $rmse=0.9335$ ,  $weight=0.0402$   
Supp-kNN on residuals of full global effects

### **Combinations:**

Each of the following results is based on mixing two individual results. Before mixing we split the user-movie pairs into 15 bins based on their support. For each bin we compute unique combination coefficients based on regression involving the Probe set.

89.  $rmse=0.8976$ ,  $weight=0.0552$   
Combination of #67 with #35
90.  $rmse=0.8876$ ,  $weight=0.1471$   
Combination of #36 with <NNMF (60 factors) adaptive user factors by MseSim>
91.  $rmse=0.8977$ ,  $weight=0.1053$   
Combination of #81 with #75
92.  $rmse=0.8909$ ,  $weight=0.0588$   
Combination of #45 with <User-kNN on all global effects but the last 4>
93.  $rmse=0.9003$ ,  $weight=0.0757$   
Combination of #50 with <NNMF (20 factors, adaptive user factors by [MseSim]>
94.  $rmse=0.8906$ ,  $weight=-0.0634$   
Combination of #45 with #73
95.  $rmse=0.9024$ ,  $weight=0.0569$   
Combination of #73 with <50 neighbors Slow-kNN on residuals of all global effects except last 4>
96.  $rmse=0.9078$ ,  $weight=0.0372$   
Combination of #84 with <User-kNN on raw scores>
97.  $rmse=0.9046$ ,  $weight=0.0508$   
Combination of #74 with #66

### **Imputation of Qualifying predictions:**

We had predictions for the Qualifying set with RMSE of 0.8836. Then, we inserted the Qualifying set into the training set, while setting unknown scores to the RMSE= 0.8836 predictions. We tried some of our methods on this enhanced training set:

98.  $rmse=0.8952$ ,  $weight=0.0937$   
MSE-kNN on residuals of SimuFctr (20 factors)
99.  $rmse=0.9100$ ,  $weight=-0.0314$   
IncFctr (40 factors)
100.  $rmse=0.9039$ ,  $weight=0.0735$   
IncFctr (40 factors), adaptive user factors by [SuppSim]
101.  $rmse=0.9056$ ,  $weight=-0.1866$   
SimuFctr (20 factors), Probe set is excluded from training set
102.  $rmse=0.9093$ ,  $weight=-0.0769$   
IncFctr (40 factors), adaptive user factors by [SuppSim]. Probe set is excluded from training set
103.  $rmse=0.9005$ ,  $weight=0.0503$   
MSE-kNN on residuals of IncFctr (40 factors)
104.  $rmse=0.8975$ ,  $weight=0.1155$   
A combination (by 15 support-base bins) of #99 with <SimuFctr (20 factors)>

## Specials:

105.  $rmse=1.1263$ ,  $weight=-0.1345$

Take binary matrix (rated=1, not-rated=0), and estimate it by 40 factors. Using this factors, construct predictions for the Probe and Qualifying set and center the predictions for each set. Consequently, using the probe set we learn how to regress centered true ratings on these predictions, and do the same on the Qualifying set.

106.  $rmse=0.9162$ ,  $weight=-0.0702$

This method fits a series of models, each using the residuals from the previous model. There were three stages of models. First, effects were fit for rating date, movie, and user. Second, interactions were fit between users and 11 movie factors. The first factor was movie effect estimated above, while the last ten factors were based on approximate principal components of the movies. The movie and user effects and all 11 interactions were shrunk using a form of empirical Bayes. Third, residuals of movies in the qualifying data set were predicted using linear combinations of residuals of correlation-based nearest neighbors. Predictions for the qualifying data equal the sums of the various models. Models from all three stages were fit using training data only (without including the Probe set).

107.  $rmse=0.9134$ ,  $weight=0.1051$

This method is similar to #106 with the following exceptions. The main effect for date of rating was excluded. After fitting main effects for movie and user, eight interactions were included for movie and user support, movie and user effects, and four versions of time (see [2], Sec. 3). For these eight interactions, the linear fits were replaced by quadratic fits, and empirical Bayes shrinkage was performed on both the linear and quadratic terms (after make them orthogonal to each other) analogously to the previous method (#106). Models for all three stages were fit using the training data plus a random 90 percent sample of the probe data.

## How many results are really needed?

For completeness, we listed all 107 results that were blended in our RMSE=0.8712 submission. It is important to note that the major reason for using all these results was convenience, as we anyway accumulated them during the year. This is the nature of an ongoing competition. However, in hindsight, we would probably drop many of these results, and recompute some in a different way. We believe that far fewer results are really necessary. For example, based on just three results one can breach the RMSE=0.8800 barrier: A blend of #8, #38, and #92, with weights 0.1893, 0.4225, and 0.4441, respectively, would already achieve a solution with an RMSE of 0.8793. Similarly, combining #8, #38, and #64 yields RMSE=0.8798. Notice that these combinations touch the main approaches (k-NN, factorization, RBMs and asymmetric factor models). In addition, we have found that at most 11 results suffice for achieving a blend with above 8% improvement over Cinematch score.

## Appendix 1 - Estimating similarity scores from binary data

Similarity scores among items are a key component within many collaborative filtering techniques. Here, we suggest a similarity measure for two items -  $i$  and  $j$  - based only on their "binary rating history". That is the identity of the users that rated them, rather than the actual ratings that they got. We have found, quite surprisingly, that in some occasions, this similarity score was more effective than a score which is based on the ratings themselves.

An obvious input to this score is  $n_{ij}$ , the number of users who viewed (or, "rated") both items.

However,  $n_{ij} = 5$  means very different things depending on whether  $n_i$  and  $n_j$ , the number of viewings of each item, are on the order of 10 each or 200 each. Consequently, some rescaling seems necessary. One option is to use  $n_{ij}/x_{ij}$ , where  $x_{ij} = n_i n_j / n$  and  $n$  is the total number of ratings. We believe that there are better choices for  $x_{ij}$ .

Consider two movies that have each been rated 10 times, but differ as follows. Movie  $j$  was always rated by someone who had rated only five other movies, while Movie  $k$  was always rated by active viewers who had each rated 100 other movies. That is, Movie  $j$  is part of 50 pairs of movies rated by the same user (including multiple occurrences), while Movie  $k$  is part of 1000 pairs. If  $n_{ij} = 3$ , that is much stronger evidence of similarity than if  $n_{ik} = 3$ .

Let  $N_i$  equal the number of pairs involving Movie  $i$ ; that is,  $N_i = \sum_{j \neq i} n_{ij}$ , and let  $N_i = \sum_i N_j$  equal twice the total number of pairs. We propose

$$x_{ij} = N_i N_j / (N - N_i) + N_i N_j / (N - N_i) .$$

This should approximately standardize  $n_{ij}$  in the sense that

$$\sum_j n_{ij} \approx \sum_j x_{ij} \quad \text{for all } i.$$

## Appendix 2 – Deriving Similarity Scores from Movie Titles [EditSim]

Some similarity information between movies can be captured from the movie titles and release year provided in the Netflix Prize data. The resulting similarity values were proved useful within the neighborhood-aware factorization. However, we doubt their utility to the overall blend. For completeness, we list below the exact formula that we used for deriving these similarity weights.

The following measure is based on our prior experience with disambiguating a user's input. We have found that a combination of plain edit distance, with an emphasis on full words and prefixes could better uncover the user's intention.

Let us concentrate on two movies, with titles  $ttl1$  and  $ttl2$ , and release years  $year1$  and  $year2$ , respectively. We use the following notation:

- $|str|$  is the length of string  $str$ .
- $\text{editD}(ttl1,ttl2)$  is the edit distance between  $ttl1$  and  $ttl2$ .
- $\text{jointPrefix}(ttl1,ttl2)$  is the longest prefix of  $ttl1$  and  $ttl2$ .
- $\text{prefixBonus}(ttl1,ttl2)$  is  $\min(5, |\text{jointPrefix}(ttl1,ttl2)|/3)$ .
- $\#\text{overlaps}(ttl1,ttl2)$  is the number of full words appearing in both  $ttl1$  and  $ttl2$ .

Our adjusted edit distance is defined as:

$$dist = \frac{\text{editD}(ttl1,ttl2)}{\max(|ttl1|, |ttl2|) \cdot \text{prefixBonus}(ttl1,ttl2)}$$

And the derived similarity score between the two movies is:

$$sim = \left( (0.2 \cdot |year1 - year2| + dist) \cdot 0.7^{\#\text{overlaps}(ttl1,ttl2)} \right)^{-0.3}$$

## Acknowledgments

We would like to thank AT&T Labs for supporting and facilitating this work. We are very lucky to work in a place that values true long term research.

We had interesting discussions with some of our competitors. Especially, we would like to thank the ML@Toronto team for making their inspiring work publicly available. To Lester Mackey from Dinosaur Planet team for his clear explanations on RBMs. To Arek Paterek for NSVD1/2. To all competitors, especially the Gravity team, for giving us a hard time and driving us to distil our techniques and seek new directions. Also, to all people that took their time for posting in the web forum. Finally, we are grateful to Netflix for putting on this amazing competition and conducting it flawlessly.

## References

1. R. Bell and Y. Koren, ``Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights'', *IEEE International Conference on Data Mining (ICDM'07)*, IEEE, 2007.

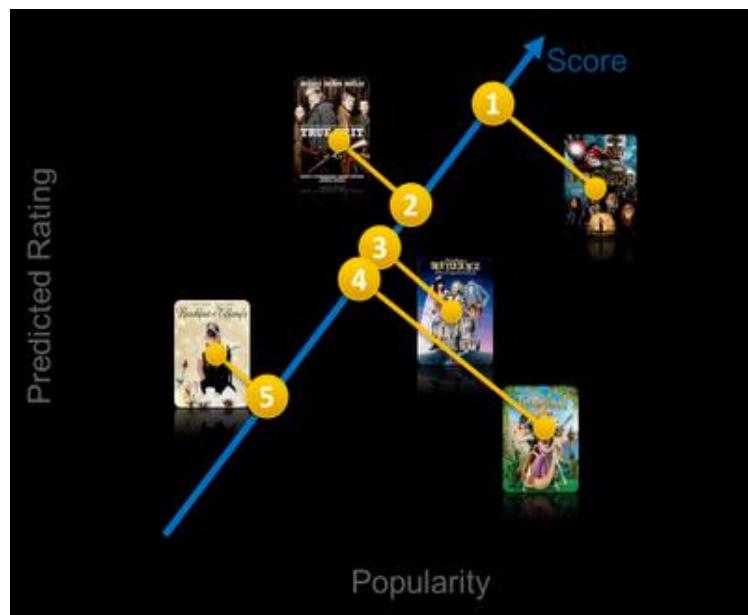
2. R. Bell and Y. Koren, ``Improved Neighborhood-based Collaborative Filtering'', *KDD-Cup and Workshop*, ACM press, 2007.
3. R. Bell, Y. Koren and C. Volinsky, ``Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems'', *Proceedings of the 13th ACM Int. Conference on Knowledge Discovery and Data Mining (KDD'07)*, ACM press, 2007.
4. A. Paterek, ``Improving regularized singular value decomposition for collaborative filtering'', *KDD-Cup and Workshop*, ACM press, 2007.
5. R. Salakhutdinov, A. Mnih and G. Hinton, ``Restricted Boltzmann machines for collaborative filtering'', *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*, 2007.
6. Y. Zhang and J. Koren, ``Efficient Bayesian Hierarchical User Modeling for Recommendation Systems'', *Proceedings of the 30st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'07)*, ACM press, 2007.

## Recommendations as Personalized Learning to Rank

As I have explained in other publications such as the [Netflix Techblog](#), ranking is a very important part of a Recommender System. Although the Netflix Prize focused on rating prediction, ranking is in most cases a much better formulation for the recommendation problem. In this post I give some more motivation, and an introduction to the problem of personalized learning to rank, with pointers to some solutions. The post is motivated, among others, by a proposal I sent for a tutorial at this year's [Recsys](#). Coincidentally, my former colleagues in Telefonica, who have been working in learning to rank for some time, proposed a very similar one. I encourage you to use this post as an introduction to [their tutorial](#), which you should definitely attend. The goal of a ranking system is to find the best possible ordering of a set of items for a user, within a specific context, in real-time. We optimize ranking algorithms to give the highest scores to titles that a member is most likely to play and enjoy.

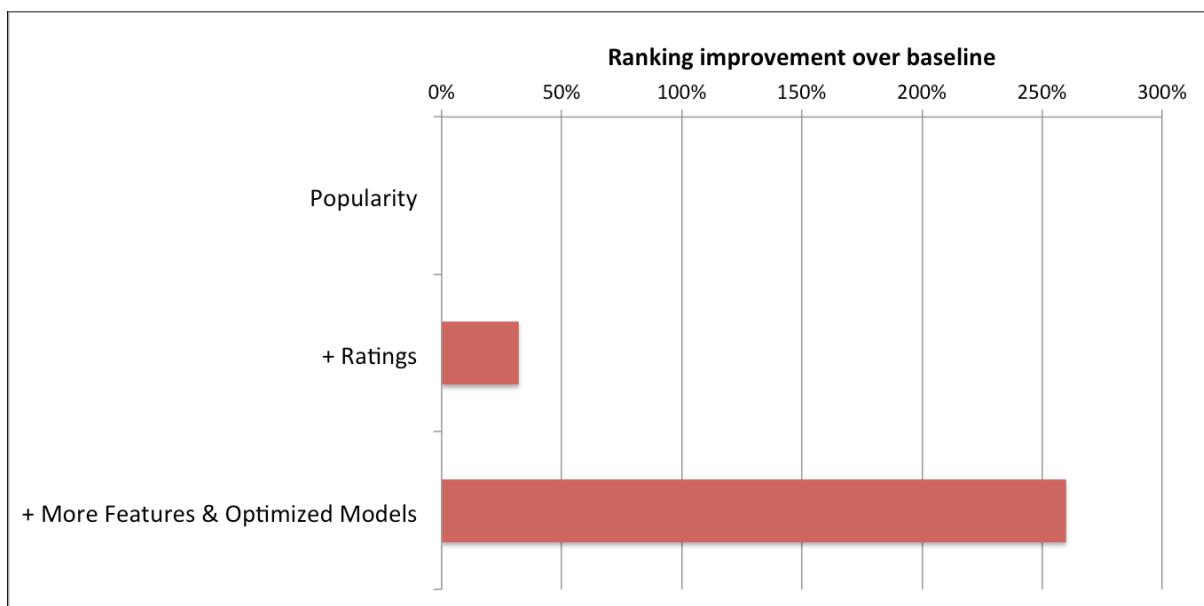
If you are looking for a ranking function that optimizes consumption, an obvious baseline is item popularity. The reason is clear: on average, a user is most likely to like what most others like. Think of the following situation: You walk into a room full of people you know nothing about, and you are asked to prepare a list of ten books each person likes. You will get \$10 for each book you guess right. Of course, your best bet in this case would be to prepare identical lists with the "10 most liked books in recent times". Chances are the people in the room is a fair sample of the overall population, and you end up making some money. However, popularity is the opposite of personalization. As I explained in the previous example, it will produce the same ordering of items for every member. The goal becomes is to find a personalized ranking function that is better than item popularity, so we can better satisfy users with varying tastes. Our goal is to recommend the items that each user is most likely to enjoy. One way to approach this is to ask users to rate a few titles they have read in the past in order to build a rating prediction component. Then, we can use the user's predicted rating of each item as an adjunct to item popularity. Using predicted ratings on their own as a ranking function can lead to items that are too niche or unfamiliar, and can exclude items that the user would want to watch even though they may not rate them highly. To compensate for this, rather than using either popularity or predicted rating on their own, we would like to produce rankings that balance both of these aspects. At this point, we are ready to build a ranking prediction model using these two features.

Let us start with a very simple scoring approach by choosing our ranking function to be a linear combination of popularity and predicted rating. This gives an equation of the form  $score(u,v) = w_1 p(v) + w_2 r(u,v) + b$ , where  $u=\text{user}, v=\text{video item}$ ,  $p=\text{popularity}$  and  $r=\text{predicted rating}$ . This equation defines a two-dimensional space as the one depicted in the following figure.



Once we have such a function, we can pass a set of videos through our function and sort them in descending order according to the score. First, though, we need to determine the weights  $w_1$  and  $w_2$  in our model (the bias  $b$  is constant and thus ends up not affecting the final ordering). We can formulate this as a machine learning problem: select positive and negative examples from your historical data and let a machine learning algorithm learn the weights that optimize our goal. This family of machine learning problems is known as "Learning to Rank" and is central to application scenarios such as search engines or ad targeting. A crucial difference in the case of ranked recommendations is the importance of personalization: we do not expect a global notion of relevance, but rather look for ways of optimizing a personalized model.

As you might guess, the previous two-dimensional model is a very basic baseline. Apart from popularity and rating prediction, you can think on adding all kinds of features related to the user, the item, or the user-item pair. Below you can see a graph showing the improvement we have seen at Netflix after adding many different features and optimizing the models.



The traditional pointwise approach to learning to rank described above treats ranking as a simple binary classification problem where the only input are positive and negative examples. Typical models used in this context include Logistic Regression, Support Vector Machines, Random Forests or Gradient Boosted Decision Trees.

There is a growing research effort in finding better approaches to ranking. The pairwise approach to ranking, for instance, optimizes a loss function defined on pairwise preferences from the user. The goal is to minimize the number of inversions in the resulting ranking. Once we have reformulated the problem this way, we can transform it back into the previous binary classification problem. Examples of such an approach are RankSVM [Chapelle and Keerthi, 2010, [Efficient algorithms for ranking with SVMs](#)], RankBoost [Freund et al., 2003, [An efficient boosting algorithm for combining preferences](#)], or RankNet [Burges et al., 2005, [Learning to rank using gradient descent](#)].

We can also try to directly optimize the ranking of the whole list by using a listwise approach. RankCosine [Xia et al., 2008. [Listwise approach to learning to rank: theory and algorithm](#)], for example, uses similarity between the ranking list and the ground truth as a loss function. ListNet [Cao et al., 2007. [Learning to rank: From pairwise approach to listwise approach](#)] uses KL-divergence as loss function by defining a probability distribution. RankALS [Takacs and Tikk. 2012. [Alternating least squares for personalized ranking](#)] is a recent approach that defines an objective function that directly includes the ranking optimization and then uses Alternating Least Squares (ALS) for optimizing.

Whatever ranking approach we use, we need to use rank-specific information retrieval metrics to measure the performance of the model. Some of those metrics include [Mean Average Precision](#) (MAP), [Normalized Discounted Cumulative Gain](#) (NDCG), [Mean Reciprocal Rank](#) (MRR), or [Fraction of Concordant Pairs](#) (FCP). What we would ideally like to do is to directly optimize those same metrics. However, it is hard to optimize machine-learned models directly on these measures since they are not differentiable and standard methods such as gradient descent or ALS cannot be directly applied. In order to optimize those metrics, some methods find a smoothed version of the objective function to run Gradient Descent. CLiMF optimizes MRR [Shi et al. 2012. [CLiMF: learning to maximize reciprocal rank with collaborative less-is-more filtering](#)], and TFMAP [Shi et al. 2012. [TFMAP: optimizing MAP for top-n context-aware recommendation](#)], optimizes MAP in a similar way. The same authors have very recently added a third variation in which they use a similar approach to optimize "graded relevance" domains such as ratings [Shi et. al, "Gapfm: Optimal Top-N Recommendations for Graded Relevance Domains"]. AdaRank [Xu and Li. 2007. [AdaRank: a boosting algorithm for information retrieval](#)] uses boosting to optimize NDCG. Another method to optimize NDCG is NDCG-Boost [Valizadegan et al. 2000. [Learning to Rank by Optimizing NDCG Measure](#)], which optimizes expectation of NDCG over all possible permutations. SVM-MAP [Xu et al. 2008. [Directly optimizing evaluation measures in learning to rank](#)] relaxes the MAP metric by adding it to the SVM constraints. It is even possible to directly optimize the non-differentiable IR metrics by using techniques such as Genetic Programming, Simulated Annealing [Karimzadehgan et al. 2011. [A stochastic learning-to-rank algorithm and its application to contextual advertising](#)], or even Particle Swarming [Diaz-Aviles et al. 2012. [Swarming to rank for recommender systems](#)].

As I mentioned at the beginning of the post, the traditional formulation for the recommender problem was that of a rating prediction. However, learning to rank offers a much better formal framework in most contexts. There is a lot of interesting research happening in this area, but it is definitely worth for more researchers to focus their efforts on what is a very real and practical problem where one can have a great impact.

Posted 24th July 2013 by [Xavier Amatriain](#)

# Tutorial on Recent Progress in Collaborative Filtering

Yehuda Koren

Collaborative filtering is a relatively young algorithmic approach, which already found its way into many commercial applications and established itself as a prime component of recommender systems. The field enjoyed a surge of interest thanks to the Netflix Prize competition that began in October 2006. Impact on research was significant in several ways. For the first time, public research community could access a large-scale industrial strong dataset. In addition, thousands of scientists, students, engineers and enthusiasts were attracted to the field. Importantly, all are judged by the same rigid yardstick, what makes comparisons and experiments much more meaningful. The nature of the competition encourages a rapid development pace, where first generation techniques were quickly replaced by second and third generation innovations.

Pushing techniques to be more and more accurate requires deepening their foundations, while reducing reliance on arbitrary decisions. An interesting outcome is forming surprising links among seemingly different techniques. For example, at their limit, user-user and item-item neighborhood models may converge to a single model. Furthermore, at that point both become equivalent to a simple matrix factorization model.

Some previous distinctions become less relevant. A major example is the traditional broad categorization of techniques into “model based” and “memory based”, which is no longer appropriate. Traditional heuristic, memory-based techniques evolved to rely on rigorous models. On the other hand, some techniques which were considered purely model-based, are now improving their accuracy by also directly accessing all past ratings, a memory-based habit.

The quest for more accurate models does not stop at pushing the foundations of the models to their limits. At least as important is the identification of which kinds of signal, or features, are extractable from the data. Conventional techniques address the sparse data of user-item preferences (or, ratings). Accuracy significantly improves by also addressing less obvious sources of information. One prime example includes all kinds of temporal effects reflecting the dynamic, time-series nature of the data. Not less important is listening to hidden feedback that users provide. One kind of such feedback is which items they chose to rate (regardless of rating values). Rated items are not selected at random, but rather reveal

interesting aspects of user preferences, going far and beyond the numerical values of the ratings.

There are many different ways to measure model accuracy. Commonly used measures, such as RMSE (or, MAE) tend to seemingly diminish differences among models. However, a new study finds that a small difference in RMSE can lead to a very significant improvement when ordering items by their predicted preferences. This way, quality of the top-K items presented to a user can vastly improve. For example, we have found that replacing traditional Pearson-coefficient item-item models with more rigorous item-item models triples the probability of placing a top-rated product at the top of a recommendation list.

However improved accuracy is not a panacea. There are several other challenges for collaborative filtering recommenders; we will deal with some of them. Obviously, as in all fields of computer science, computational efficiency is a key consideration. Some recommender systems need to scale to millions of users and items placing a high premium on efficiency. Explainability is another key aspect of recommenders. Users will likely benefit more from recommendations when they are accompanied with an intuitive reasoning such as “you will like this movie because you liked those movies”. Good explainability is not going hand in hand with accuracy. In reality, it may be challenging to explain the reasoning beyond the more sophisticated approaches. Yet, much can be done to bridge these two important goals and devise accurate and explainable techniques. Another area that attracts new developments is the analysis of implicit feedback. While most research is centered on *explicit feedback*, where users tell us about their preferences, in many scenarios such feedback is hard to collect and we need to work with the more prevalent *implicit feedback*. Then, unobtrusive observations on users’ behavior hint us on their preferences. Beyond the obvious difficulties in collecting and interpreting implicit feedback, it also requires special treatment through revamped models. To begin with, reliable negative feedback is no longer available. The sparse set of observed actions corresponds to only positive feedback, making it necessary to look at the broader full dense set of user-item relations.

To summarize, we believe that collaborative filtering is still a young field with many open challenges, which we, researchers, like to view as opportunities. Recent progress, mainly thanks to the widely popular Netflix Prize competition led to some innovations that will hopefully mark a significant impact on the field and find their way into the growing number of real life recommender systems.

## Possibly Wrong

*On science, mathematics, and computing*

---

## Reddit's comment ranking algorithm

Posted on June 5, 2011

On a web site where users rate the quality of—well, just about anything, from consumer products to comments on news articles—with positive or negative “votes,” how should the rated items be sorted so as to present the higher-rated items near the top of the page and the lower-rated items near the bottom?

I recently learned how Reddit handles this problem when sorting user comments on a link post. (I was teased to the [Reddit link](#) by the title’s suggestion that Randall Munroe of [xkcd](#) fame had a hand in the algorithm.) There is some interesting mathematics involved...but there also appear to be some equally interesting bugs in the implementation, which I will discuss here.

[Edit: Two of the three problems discussed here have been fixed.]

First, some background, in chronological order: Evan Miller has a great write-up titled “[How Not To Sort By Average Rating](#).” He shows why a few simple solutions do not work very well, and instead proposes using the lower bound of the [Wilson score](#) confidence interval to sort ratings, including Ruby source code implementing the formula.

Evan’s post was in turn referenced by a Reddit blog [guest post](#) by Randall, describing Reddit’s new “best” comment ranking algorithm around the time it was implemented. And finally, the most recent [amix blog](#) by Amir Salihefendic describes Reddit’s ranking algorithms in excellent detail, including inspection of the Reddit source code. (To be clear, Reddit uses different ranking algorithms for “stories” (i.e., link posts) vs. user comments. We are focusing on the latter here.)

I had initially planned to discuss confidence intervals in general, and some interesting issues with binomial proportion estimation in particular, of which the Wilson score interval is just one example. There frequently seems to be confusion about just what information is conveyed by a confidence interval. For example, Evan describes the Wilson score lower bound as the answer to the question, “*Given the ratings I have, there is a 95% chance that the “real” fraction of positive ratings is at least what?*” This is misleading, since no claim can be made about the probability that the “real” fraction of positive ratings lies within a particular computed interval. (Wikipedia actually does a great job of [explaining this](#).)

But let’s skip that and instead focus on the actual implementation in the Reddit [source code](#). I’ll take Amir’s approach of presenting a Python translation, since the original is in the less common [Pyrex](#):

```

1 # From /r2/r2/lib/db/_sorts.pyx
2 from math import sqrt
3
4 def confidence(ups, downs):
5     n = ups + downs
6     if n == 0:
7         return 0
8     z = 1.0 #1.0 = 85%, 1.6 = 95%
9     phat = float(ups) / n
10    return sqrt(phat+z*z/(2*n)-z*((phat*(1-phat)+z*z/(4*n))/n))/(1+z*z/n)

```

Compare this with the formula for the endpoints of the Wilson score interval:

$$\frac{\hat{p} + \frac{z^2}{2n} \pm z \sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}$$

where  $\hat{p}$  is the fraction of observed votes that are positive, and  $z$  is the appropriate quantile of the standard normal distribution (more on this shortly).

We can make several observations at this point. First, and most importantly, the square root is in the wrong place in the code! I am not sure how this error crept in, since the form of the expression and even the variable names are nearly identical to the correct Ruby example from Evan's original post. At any rate, whatever Reddit is computing, it isn't the Wilson score interval.

Second, it is worth clarifying for exactly what confidence interval we are trying to compute endpoints. Both Evan and Amir present the above formula with the plus/minus notation, and use  $\alpha/2$  in their  $z$ -subscripts, which together strongly suggest that the *two-sided* Wilson score interval of the form  $[c - \Delta, c + \Delta]$  is intended. However, the *text* in Evan's post and the actual constants used for  $z$  in the code comments suggest a *one-sided* interval of the form  $[u, 1]$ , where  $u$  is the value returned by the function. Since the distinction only matters when we talk about the actual level of confidence in the resulting interval (e.g., is it 95% or 90%, 85% or 70%, etc.), to avoid further confusion I will use the code as a guide and refer to levels of confidence assuming *one-sided* intervals.

Finally, one interesting side effect of the current incorrect implementation is that the function always returns positive values, *even for zero upvotes*. More generally, the resulting interval frequently does not even contain the maximum likelihood estimate of the true fraction of positive ratings! That's the bad news... but the good news is that the function "does the right thing" for zero upvotes anyway, by returning smaller values for larger numbers of *downvotes*. The true Wilson score interval lower bound does not have this nice property, since the lower bound is zero for  $\hat{p} = 0$ , independent of  $n$ .

(Actually, I am not sure if this is ever a problem, at least on Reddit, since every comment seems to begin its life with one "automatic" upvote.)

So, how should we fix things? I would recommend the following modification to the algorithm:

```

1 # From /r2/r2/lib/db/_sorts.pyx
2 from math import sqrt
3
4 def confidence_fixed(ups, downs):
5     if ups == 0:
6         return -downs
7     n = ups + downs
8     z = 1.64485 #1.0 = 85%, 1.6 = 95%
9     phat = float(ups) / n
10    return (phat+z*z/(2*n)-z*sqrt((phat*(1-phat)+z*z/(4*n))/n))/(1+z*z/n)

```

This addresses each of the three issues described above:

First, we fix the square root in the formula, so that we are indeed computing the lower bound of the Wilson score interval (usually, anyway; see below).

Second, we change the constant  $z$  to use the 95% confidence interval instead of the original 85%. This is really just a fine-tuning to keep the resulting ordering "closer," in an eyeball-norm sense, to how the current algorithm performs.

Finally, we extend the special-case behavior from just returning zero for zero *total* votes, to returning *negative* values for zero *upvotes*. This maintains the current property that more downvotes will always push items farther down the list.

**Share this:**

Be the first to like this.

**Related**

[Reddit's comment ranking algorithm revisited](#)  
With 2 comments

[Puzzle: Randomly shuffling songs in iTunes](#)  
With 3 comments

[Card shuffling algorithms, good and bad](#)  
With 2 comments

This entry was posted in [Uncategorized](#). Bookmark the [permalink](#).

## 13 Responses to *Reddit's comment ranking algorithm*

Pingback: [Reddit: ¿qué sistema usa para puntuar las entradas?](#)

Pingback: [Reddit's ACTUAL story ranking algorithm explained \(significant typos in previously published version\) | Bibliographic Wilderness](#)

 **john** says:  
May 14, 2012 at 5:52 am

The June 2010 version of reddit's code has the bug you highlight, but the April 2011 version has fixed it and clarified the code a bit!

[https://github.com/reddit/reddit/commits/master/r2/r2/lib/db/\\_sorts.pyx](https://github.com/reddit/reddit/commits/master/r2/r2/lib/db/_sorts.pyx) and click through to the two versions.

[Reply](#)

---

 **possiblywrong** says:  
May 14, 2012 at 1:08 pm

Thanks for pointing this out, John. I edited this post accordingly to note these fixes.

There remains the question of the third “problem” described in the post. That is, the fixed version always returns a confidence bound of 0 for 0 upvotes, independent of the number of downvotes. It seems like it would be desirable for more downvotes to (strictly) decrease the confidence score, even for upvotes=0. (Both the old buggy version, and my suggested fix, had this property.)

But maybe this situation never arises, since comments always seem to start with 1 upvote?

[Reply](#)

---

 **ajkj** says:  
October 26, 2012 at 2:47 am

It's possible to remove that 1 upvote from your own comment if you want, so it can happen.

---

 **Wesley** says:  
August 11, 2013 at 1:59 am

Shouldn't the z-score be 1.96 for a 95% interval? 1.64485 should be for 90%?

[Reply](#)

**possiblywrong** says:

August 11, 2013 at 11:43 pm

This has to do with the “two-sided” vs. “one-sided” issue discussed in the post. That is, do we want an interval with two endpoints (strictly between 0 and 1), so that 95% of the time the interval will contain the “true” probability of an up-vote? Or do we just want a \*lower bound\* that 95% of the time will be less than the true probability of an up-vote?

If it’s the former (as the other referenced blog posts suggest), then you’re right, typical practice is to split the 5% equally between the upper and lower tails. But if it’s the latter (as the actual Reddit code comments suggest), then we want an interval of the form  $(a, 1]$ , and the lower tail gets the whole 5%.

Of course, this is all simply an approximation that is meant to \*sort\* comments in a relative sense, not measure anything \*absolutely\* for any one comment. So it really doesn’t matter— that is, we can choose to interpret the “score” either way: it is either (1) the lower of two endpoints of an interval that contains the true probability 90% of the time, or (2) the value  $a$  such that the interval  $(a, 1]$  contains the true probability 95% of the time.

[Reply](#)**NK** says:

May 21, 2014 at 6:20 pm

Have you come across an approach that also takes into account the age of the positive and negative ratings?

**possiblywrong** says:

May 22, 2014 at 5:15 pm

@NK: If I understand your question correctly, the “hot” ranking of comments (see the same source code link in the post) attempts to do this, or at least, it generally degrades a comment’s ranking over time (relative to newer ones).

[Reply](#)**howard** says:

November 26, 2014 at 3:05 am

i have a question regarding how the confidence sort rank a message with 0 up and 0 down votes (no vote at all) compare to a message with 0 upvote and 1 downvote.

using the the confidence sort it will rank the message with 0 upvote and 1 downvote higher than message with 0 up and down vote. this doesn’t seem to make sense to me.

am i missing something?

[Reply](#)**possiblywrong** says:

November 26, 2014 at 1:26 pm

[Follow](#)

Hmmm. I think it

confidence scoring implementation you are looking at:

### Follow “Possibly Wrong”

1. If you are looking for a “hot” place”), then you

implementation quoted in this post (with the square root “in the wrong place”). The simpler incorrect implementation is that a comment with (0 up, 1 down) has a higher score than a comment with (1 up, 0 down).

Get every new post delivered to your Inbox.

Join 41 other followers

2. Note, however, that a comment always has a higher score than any other comment with 0 upvotes. This is arguably not an improvement, IMHO.

Enter your email address

[Sign me up](#)

3. My suggested fix is to rank comments with 0 upvotes so that  $(0,0) > (0,1) > (0,2) > \dots$ . That is, with zero upvotes, more downvotes are always worse.

If I understand you correctly, we agree that (3) is the “right” behavior, at least for comments with 0 upvotes. There is further argument for this being the “right” ordering in a [follow-up post](#) proposing what is IMO a less ad hoc ordering that yields this same behavior.

[Reply](#)



**howard** says:

December 1, 2014 at 2:28 pm

possiblywrong, thanks for your reply

I have also noticed that  $(1,X)$  will always rank higher than  $(0,0)$  even if  $X$  is a huge number for example  $(1,1000)$ . how did you resolve this? does this make sense?



**possiblywrong** says:

December 1, 2014 at 3:23 pm

@howard, this is a tricky one, since ranking a comment with absolutely no votes depends a lot on your assumptions about “prior” knowledge of the value of a comment. You’re right that Reddit’s current implementation ranks  $(0,0) < (1,x)$ . But note that my proposed ranking in the follow-up post mentioned earlier handles it differently, ranking  $\dots < (1,3) < (1,2) < (0,0) < (1,1) < (1,0)$ . So there is definitely an argument for other interpretations/assumptions.

[Reply](#)

## Possibly Wrong

*The Twenty Ten Theme.* [Blog at WordPress.com.](#)

# Lessons from the Netflix Prize Challenge

Robert M. Bell and Yehuda Koren  
AT&T Labs – Research  
180 Park Ave, Florham Park, NJ  
[{rbell,yehuda}@research.att.com](mailto:{rbell,yehuda}@research.att.com)

## ABSTRACT

This article outlines the overall strategy and summarizes a few key innovations of the team that won the first Netflix progress prize.

## 1. INTRODUCTION

In October 2006, Netflix Inc. released more than 100 million customer generated movie ratings as part of the Netflix Prize competition. The goal of the competition is to produce a 10 percent reduction in the root mean squared error (RMSE) of test data, relative to the RMSE achieved by Cinematch, the technology that currently produces movie recommendations for Netflix customers. A prize of \$1,000,000 will be awarded to the first team to reach that goal [5]. These data and the prize have generated unprecedented interest and advancement in the field of collaborative filtering, a class of methods that analyze past user behavior to infer relationships among items and to inform item recommendations for users. Many of these advances have been shared, most notably in the Netflix Prize forum [8] and a 2007 KDD workshop [1].

Three characteristics of the Netflix data combine to pose a large challenge for prediction. First and most obvious is size. Netflix published a comprehensive data set including more than 100 million movie ratings that were performed by about 480,000 users on 17,770 movies. This places a premium on methods that can be computed efficiently. Second, almost 99% of the potential user-item pairs have no rating. Consequently, machine learning methods designed for complete data situations, or nearly so, must be modified or abandoned. Third, the pattern of observed data is very nonrandom; i.e., the amount of observed data varies by more than three orders of magnitude among users or among items. This fact complicates the challenge to detect weak signals for users/movies with sufficient sample size while avoiding over fitting for users/movies with very few ratings.

To further complicate matters, peoples' taste for movies is a very complex process. A particular user's rating of a movie might be affected by any of countless factors ranging from the general—e.g., genre—to quite specific attributes such as the actors, the setting, and the style of background music. Simultaneously modeling such a wide spectrum of factors adds to the challenge.

While no team had reached the 10 percent improvement level after one year, Netflix awarded the first annual progress prize to the KorBell team of AT&T Labs-Research, which

achieved the greatest improvement at that time, 8.43%. This article describes some of the strategies that contributed to that team's success.

First, it was important to utilize a variety of models that complement the shortcomings of each other. In particular, this includes both nearest neighbor models (k-NN) and latent factor models such as SVD/factorization or restricted Boltzmann machines. In addition, it was important to include models that incorporated information beyond the ratings themselves—e.g., *what* movies a particular user rated. Second, we developed several innovations that improved existing collaborative filtering methods, notably:

- A new method for computing nearest neighbor interpolation weights that better accounts for interactions among neighbors.
- A neighborhood-aware factorization method that improves standard factorization models by optimizing criteria more specific to the targets of specific predictions.
- Integration of information about which movies a user rated into latent factor models for the ratings themselves, adapting techniques from [9; 10].
- New regularization methods across a variety of models, including both neighborhood and latent factor models.

Section 2 discusses the need to utilize multiple, complementary models. Section 3 summarizes a few of the innovations developed in the process.

## 2. UTILIZING A COMPLEMENTARY SET OF MODELS

We found no perfect model. Instead, our best results came from combining predictions of models that complemented each other. While our winning entry, a linear combination of many prediction sets, achieved an improvement over Cinematch of 8.43%, the best single set of predictions reached only 6.57%. Even that method was a hybrid based on applying neighborhood methods to results of a restricted Boltzmann machine. The best improvement achieved purely by a latent factor model was 5.10% and was even less for the best pure nearest neighbor method.

We found that it was critically important to utilize a variety of methods because the two main tools for collaborative filtering—neighborhood models and latent factor models—address quite different levels of structure in the data.

Neighborhood models (k-NN)—the most common form of collaborative filtering—are most effective at detecting very

localized relationships. The item neighborhood approach identifies pairs of items (movies) that tend to be rated similarly, in order to predict ratings for an unrated item based on ratings of similar (neighboring) items by the same user [11]. Similarly, the user approach bases predictions on ratings of the same item by similar users [7]; however, the rest of this article considers only the item approach. Our best neighborhood models typically used 20 to 50 neighboring movies for any single prediction, often ignoring the vast majority of ratings by the user of interest [2]. Consequently, these methods are unable to capture the totality of weak signals encompassed in all of a user's ratings.

Latent factor models comprise an alternative approach to collaborative filtering with the more holistic goal to uncover latent features that explain the observed ratings. For an  $m \times n$  ratings matrix  $R$  with no missing values, matrix factorization via singular value decomposition (SVD) approximates  $R$  with the best rank- $f$  approximation  $R^f$ , defined as the product of two rank- $f$  matrices  $P_{m \times f}$  and  $Q_{n \times f}$ , where  $f$  is usually much smaller than either  $m$  or  $n$ . The matrix  $R^f$  captures the  $f$  most prominent features of the data, leaving out less significant patterns in the observed data that might be mere noise. For sparse data like the Netflix ratings, alternative estimation methods are required in order to deal with the missing elements and to avoid over fitting (see, e.g., [3; 4; 6; 10]). Restricted Boltzmann machines provide another model based on estimating latent factors [10].

Latent factor models are generally effective at estimating overall structure that relates simultaneously to most or all movies. However, these models are poor at detecting strong associations among a small set of closely related movies such as The Lord of the Rings trilogy, precisely where neighborhood models do best.

While, in theory, either methodology might be extended to encompass the full spectrum of relationships, such an effort seems doomed to failure in practice. Finding niche clusters of items such as Hitchcock films or boxing movies might require thousands of latent factors. And even so, those factors might be swamped by the noise associated with the thousands of other movies. Similarly, for neighborhood methods to capture the degree that a user prefers action to dialogue may require a very large number of neighbors, to the exclusion of estimating the user's interest in other relevant features.

We also found it extremely valuable to utilize models that incorporate information beyond simply the ratings themselves. This was important for better modeling the users, as most users in the test set provided a meager number of ratings. In pursuing this approach, we analyzed *which* movies users rate, regardless of *how* they rated these movies. Hence, we try modeling for which movie a user will choose to voice his/her opinion and vote a (positive or negative) rating. This provided an important tool for learning about users, generating a unique perspective that nicely complemented the ratings-based information. We call this "the binary view of the data". Overall, this was a key to our progress, and will probably serve an even bigger role in real life recommender systems, where richer and comprehensive implicit user behavior (e.g., rental history, browsing history and search patterns) is available to enrich models centered around explicit ratings.

Finally, some of our models utilized other information about items, users, or individual ratings such as the number of rat-

ings of an item or by a user, the average rating of an item or by a user, and the date of the rating (see section on "Global Effects" in [2] for details). Variables like these allowed us, for example, to distinguish users who like the most commonly rated movies best from those who prefer more specialized fare, or to predict better for users whose ratings rise over time, above and beyond any change explained by the inherent quality of the items being rated.

### 3. IMPROVING EXISTING METHODS

Generally speaking, improved modeling tends to follow one of three directions, or a combination thereof:

1. Deepening known methods in order to improve their quality
2. Combining multi-scale views of the data
3. Combining explicit rating information with implicit rating behavior (the binary view)

Accordingly, we will demonstrate our improvements on two well known models.

#### 3.1 Weights for Neighborhood Models

Our goal is to predict an unobserved rating by user  $u$  for item (movie)  $i$ , denoted as  $r_{ui}$ . An item-oriented neighborhood model identifies a set of neighboring items  $N(i; u)$  that other users tend to rate similarly to their rating of  $i$ . All items in  $N(i; u)$  must have been rated by  $u$ . The predicted value of  $r_{ui}$  is taken as a weighted average of the ratings of neighboring items:

$$\hat{r}_{ui} \leftarrow b_{ui} + \frac{\sum_{j \in N(i; u)} s_{ij} (r_{uj} - b_{uj})}{\sum_{j \in N(i; u)} s_{ij}} \quad (1)$$

The item similarities—denoted by  $s_{ij}$ —play a central role here, as they are used both for selecting the neighbors and for weighting the above average. Common choices are the Pearson correlation coefficient and the closely related cosine similarity. We use  $b_{ui}$  to denote some baseline predictor for  $r_{ui}$ . It is important to use these baseline values to remove item- and user-specific biases that may prevent the model from revealing the more fundamental relationships. Prior methods often take  $b_{ui}$  as the mean rating of user  $u$  or item  $i$ . In [2; 3] we offered a more comprehensive approach to these baseline estimates.

Neighborhood-based methods became very popular because they are intuitive and relatively simple to implement. In particular, they do not require tuning many parameters or an extensive training stage. They also provide a concise and intuitive justification for the computed predictions. However, standard neighborhood-based methods raise some concerns:

1. The similarity function  $s_{ij}$ , which directly defines the interpolation weights, is arbitrary. Various algorithms use somewhat different similarity measures, trying to quantify the elusive notion of item similarity. Suppose that a particular item is predicted perfectly by a subset of the neighbors. In that case, we would want the predictive subset to receive all the weight, but that is impossible for bounded similarity scores like the Pearson correlation coefficient.

2. Previous neighborhood-based methods do not account for interactions among neighbors. Each similarity between an item  $i$  and a neighbor  $j \in N(i; u)$  is computed independently of the content of  $N(i; u)$  and the other similarities:  $s_{ik}$  for  $k \in N(i; u) - \{j\}$ . For example, suppose that our items are movies, and the neighbors set contains three movies that are highly correlated with each other (e.g., sequels such as “Lord of the Rings 1–3”). An algorithm that ignores the similarity of the three movies when determining their interpolation weights, may end up essentially triple counting the information provided by the group.
3. By definition, the interpolation weights sum to one, which may cause overfitting. Suppose that an item has no useful neighbors rated by a particular user. In that case, it would be best to ignore the neighborhood information, staying with the current baseline estimate. Nevertheless, the standard neighborhood formula uses a weighted average of ratings for the uninformative neighbors.
4. Neighborhood methods may not work well if variability differs substantially among neighbors.

We propose a method that overcomes these difficulties. First, we replace the weighted average by a more general weighted sum, which allows downplaying neighborhood information when lacking informative neighbors. Given a set of neighbors  $N(i; u)$  we need to compute *interpolation weights*  $\{w_{ij} | j \in N(i; u)\}$  that will enable the best prediction rule of the form:

$$\hat{r}_{ui} \leftarrow b_{ui} + \sum_{j \in N(i; u)} w_{ij}(r_{uj} - b_{uj}) \quad (2)$$

We learn interpolation weights by modeling the relationships between item  $i$  and its neighbors through a least squares problem:

$$\min_w \sum_{v \neq u} \left( r_{vi} - b_{vi} - \sum_{j \in N(i; u)} w_{ij}(r_{vj} - b_{vj}) \right)^2 \quad (3)$$

The major challenge is to cope with the missing values, and this can be done efficiently by estimating all inner products between movie ratings. For a full description refer to [3]. Importantly, this scheme provides interpolation weights that are derived directly from the ratings residuals ( $r_{uj} - b_{uj}$ ), not based on any similarity measure. Moreover, derivation of the interpolation weights explicitly accounts for relationships among the neighbors.

Our experiments showed that this scheme significantly improved accuracy relative to that based on more standard interpolation weights (e.g., driving RMSE from around 0.94 to around 0.92 for Netflix’s probe data) without a meaningful increase of running time [3].

So far, we demonstrated the benefits of building a deeper, fundamental model for k-NN. Additional accuracy gains are possible by combining the local scale view of k-NN with the higher scale view of latent factor model. Accordingly, we can take the baseline estimates (the  $b_{ui}$ ’s) as the predictions made by a latent factors model, what amounts to activating k-NN on the residuals of factorization. This enabled us to achieve RMSE of 0.898 for the test data, benefiting from the multi-scale view of the data. Further improvement is achieved by integrating also the “binary viewpoint”.

One way to achieve this is by utilizing conditional restricted Boltzmann machines (RBM) [10]. The RBM model achieves a similar accuracy to standard factorization model, while relying also on the binary viewpoint. When applying k-NN on residuals of RBM, thereby integrating all important viewpoints of the data (local, high scale and binary), RMSE dropped to 0.889.

### 3.2 Latent Factor Models

Latent factor models measure the agreement of users and movies across a series of features that are algorithmically learned from the data. This way, we associate each user  $u$  with a user-factors vector  $p_u \in \mathbb{R}^f$ , and each movie with a movie-factors vector  $q_i \in \mathbb{R}^f$ . The prediction is done by taking an inner product  $-\hat{r}_{ui} = p_u^T q_i$ . The more involved part is the estimation of the factors. A straightforward approach would minimize the regularized cost function:

$$\sum_{(u,i) \in \mathcal{K}} (r_{ui} - p_u^T q_i)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2) \quad (4)$$

The set  $\mathcal{K}$  contains all  $(u, i)$ -pairs for which  $r_{ui}$  is known. The regularization parameter  $\lambda$  prevents overfitting; a typical value is  $\lambda = 0.05$ . We minimized (4) by an alternating least squares scheme (Sec. 5.1 of [3]). Others [6; 10] used gradient based techniques. When using 60 factors ( $f = 60$ ), this basic factor model predicts the test set with RMSE 0.908.

Interestingly, this basic factorization model can benefit from moving along each of the three possible directions outlined in the beginning of the section. First, we can improve accuracy by deepening the foundation of the model. The squared penalty used in (4) implicitly assumes that all factors are drawn from independent normal distributions with zero mean and the same variance. This is a quite limiting assumption. A richer model assumes a general multivariate normal distribution for the factors. Accordingly, the ratings can be modeled by the following. First, we draw user and movie factors from the joint normal distribution:

$$p_u \sim N(\mu, \Sigma) \quad q_i \sim N(\gamma, \Lambda) \quad (5)$$

Then, each rating  $r_{ui}$  is taken from the normal distribution:

$$r_{ui} = N(p_u^T q_i, \epsilon)$$

The priors given in (5) avoid overfitting by shrinking the factors towards baseline values when not enough ratings are available. Removing the parameter restrictions implied by (4) produces significant accuracy gains. For a 60 factors example, the RMSE on the test data fell to 0.899.

Naturally the added expressive power of a richer Gaussian prior comes with an added complexity of the optimization algorithms. We employed two techniques. One is based on Gibbs sampling, and the other is based on expectation maximization, where we alternate between fixing the factors and fixing the parameters; see [12] for a related technique. Alternatively, we can stay within the convenient simple formulation (4) and significantly improve prediction accuracy by integrating either the localized viewpoint or the binary viewpoint into the model. Integrating the local viewpoint gives us a more complete multi-scale view of the data. To this end we use *adaptive user factors* that model the behavior of a user within the neighborhood of the given movie; see [4]. Technically, we allow user factors to change according to

the item that is being predicted. Therefore, when predicting  $r_{ui}$  we first compute a vector  $p_u(i) \in \mathbb{R}^f$  – depending on both  $u$  and  $i$  – and then predict  $r_{ui}$  as  $(p_u(i))^T q_i$ . The computation of  $p_u(i)$  is done by tilting the squared error (4) to overweight those movies similar to  $i$ , obtaining the problem:

$$\min_{p_u(i)} \sum_{j \in N(u)} s_{ij} (r_{uj} - p_u(i)^T q_j)^2 + \lambda \|p_u(i)\|^2 \quad (6)$$

Here,  $N(u)$  is the set of movies rated by user  $u$ . The constant  $s_{ij}$  is a similarity measure between movies  $i$  and  $j$ . By integrating the local viewpoint, prediction accuracy is significantly improved. For the 60 factors case, the error drops to RMSE=0.897.

Instead of integrating the local viewpoint, we can integrate the binary viewpoint. Here we adopt a principle from Paterik's NSVD method [9]. The NSVD model refrains from explicitly parameterizing each user, but rather models each user based on the movies that he/she rated. This way, each movie  $i$  is associated with two movie-factors vectors  $q_i$  and  $x_i$ . The representation of a user  $u$  is through the weighted sum:  $(\sum_{j \in N(u)} x_j) / \sqrt{|N(u)|}$ , so  $r_{ui}$  is predicted as:  $q_i^T (\sum_{j \in N(u)} x_j) / \sqrt{|N(u)|}$ . Importantly, NSVD models user behavior purely based on the binary viewpoint. This allows a straightforward integration of the binary viewpoint into the basic factorization model. A rating is modeled by

$$\hat{r}_{ui} = q_i^T \left( p_u + \left( \sum_{j \in N(u)} x_j \right) / \sqrt{|N(u)|} \right) \quad (7)$$

The parameters are estimated by gradient descent minimizing of the associated regularized squared error. The model used with 60 factors predicts the test set with RMSE lower than 0.897.

To summarize, the accuracy of the latent factors model can be improved by following three distinct directions. First, one can deepen the foundations of the model, by assuming a more flexible and realistic probabilistic model. Alternatively, one can retain the simple structure of the original model, but gain even more accuracy by introducing complementing perspectives of the data into the model, being either the local view or the binary view.

### 3.3 Regularization

Regularization plays a central role in all of our methods. All the models described above include massive numbers of parameters, many of which must be estimated based on a small number of ratings (or pairs of ratings). Consequently, it is critically important to avoid over fitting the training data. We use shrinkage to improve the accuracy of similarity scores used for nearest neighbor selection [4]. Shrinkage of estimated inner products is essential to successful estimation of interpolation weights for the nearest neighbor model described above [2]. Our factorization models typically use ridge regression for regularization. Removal of global effects used empirical Bayes shrinkage [2].

## 4. DISCUSSION

Our experience with the Netflix competition showed that the most successful model is an ensemble of multiple predictors; each of them specializes in addressing a different aspect of the data. In particular, most improvement stems

from moving models along three different axes. The first (and most obvious) axis denotes improving the quality of the models by basing them on deeper foundations. The second axis spans a multi-scale modeling of the data that integrates the localized perspectives expressed within limited neighborhoods with the more regional view expressed by latent factors models. The third axis introduces the “binary view” of user behavior that models which movies the users are likely to rate regardless of the value of the rating. This binary view can be explored by working with a compatible binary representation of the data or, alternatively, by employing models that allow expressing binary aspects within the context of ratings data. These models include conditional restricted Boltzmann machines<sup>1</sup>[10], NSVD [9], and some related specialized models.

Although the winning Netflix entry combined many sets of predictions, we note that it is possible to achieve a 7.58% improvement with a linear combination of three prediction sets that combine all the key elements discussed in this article: k-NN models, removal of global effects, neighborhood aware factorization, and latent factor models with the binary viewpoint.

Real life recommender systems are exposed to two types of user feedback. One is the high quality explicit rating, where users clearly express their opinion on a product. However, this kind of explicit feedback may be difficult to collect due to system constraints and reluctance of users to cooperate. A second source of information is the abundant implicit feedback, which may be purchase/rental history, browsing patterns, search keywords, etc. Typically, a recommender system capitalizes on only one of these two sources of information. We believe that the “binary view” serves as a proxy for general implicit feedback, suggesting that a combined system utilizing both explicit and implicit input is most desirable. While the Netflix data allowed us to use only the “who rated what” information, a real life system can exploit the much broader “who rented what” information, together with all other sorts of implicit feedback. This implies that our very positive experience of integrating the implicit binary view into our models can be leveraged by real life systems that access wide ranging implicit feedback. The most challenging part of the Netflix data proved to be the many users who provided very few ratings. It is very likely that many of those users do have a rich rental history, which could allow a combined recommender system to make accurate, personalized recommendations for them, or even for customers who provided no ratings at all.

## 5. REFERENCES

- [1] ACM SIGKDD, “KDD Cup and Workshop 2007”, [www.cs.uic.edu/~liub/Netflix-KDD-Cup-2007.html](http://www.cs.uic.edu/~liub/Netflix-KDD-Cup-2007.html).
- [2] R. M. Bell and Y. Koren, “Improved Neighborhood-based Collaborative Filtering”, *Proc. KDD-Cup and Workshop at the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.

<sup>1</sup>While conditional RBMs provide a tool for exploiting additional information within Netflix’s test set [10], we believe that their real strength is efficient integration of the implicit binary view within the explicit ratings. In our tests, most improvement over non-conditional RBMs was realized without including the implicit information in Netflix’s test set.

- [3] R. M. Bell and Y. Koren, “Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights”, *Proc. IEEE International Conference on Data Mining (ICDM’07)*, 2007.
- [4] R. M. Bell, Y. Koren and C. Volinsky, “Modeling Relationships at Multiple Scales to Improve Accuracy of Large Recommender Systems”, *Proc. 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.
- [5] J. Bennett and S. Lanning, “The Netflix Prize”, *Proc. KDD-Cup and Workshop at the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.
- [6] S. Funk, “Netflix Update: Try This At Home”, <http://sifter.org/~simon/journal/20061211.html>, 2006.
- [7] J. L. Herlocker, J. A. Konstan, A. Borchers and J. Riedl, “An Algorithmic Framework for Performing Collaborative Filtering”, *Proc. 22nd ACM SIGIR Conference on Information Retrieval*, pp. 230–237, 1999.
- [8] Netflix, Inc., “Netflix Prize Forum”, [www.netflixprize.com//community/](http://www.netflixprize.com//community/).
- [9] A. Paterek, “Improving Regularized Singular Value Decomposition for Collaborative Filtering”, *Proc. KDD-Cup and Workshop at the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2007.
- [10] R. Salakhutdinov, A. Mnih and G. Hinton, “Restricted Boltzmann Machines for Collaborative Filtering”, *Proc. 24th Annual International Conference on Machine Learning (ICML’07)*, 2007.
- [11] B. Sarwar, G. Karypis, J. Konstan and J. Riedl, “Item-based Collaborative Filtering Recommendation Algorithms”, *Proc. 10th International Conference on the World Wide Web*, pp. 285-295, 2001.
- [12] Y. Zhang and J. Koren, “Efficient Bayesian Hierarchical User Modeling for Recommendation Systems”, *Proc. 30th International ACM Conference on Research and Development in Information Retrieval (SIGIR’07)*, 2007.



Universidad Autónoma de Madrid  
Escuela Politécnica Superior  
Departamento de Ingeniería Informática

# Recommender System Performance Evaluation and Prediction: An Information Retrieval Perspective

Dissertation written by  
Alejandro Bellogín Kouki  
under the supervision of  
Pablo Castells Azpilicueta  
and  
Iván Cantador Gutiérrez

Madrid, October 2012



**PhD thesis title:** Recommender System Performance Evaluation and Prediction:  
An Information Retrieval Perspective

**Author:** **Alejandro Bellogín Kouki**

**Affiliation:** Departamento de Ingeniería Informática  
Escuela Politécnica Superior  
Universidad Autónoma de Madrid, Spain

**Supervisors:** **Pablo Castells Azpilicueta**  
Universidad Autónoma de Madrid, Spain

**Iván Cantador Gutiérrez**  
Universidad Autónoma de Madrid, Spain

**Date:** October 2012

**Committee:** **Alberto Suárez González**  
Universidad Autónoma de Madrid, Spain

**Gonzalo Martínez Muñoz**  
Universidad Autónoma de Madrid, Spain

**Arjen de Vries**  
Delft University of Technology, The Netherlands

**Jun Wang**  
University College London, United Kingdom

**Lourdes Araújo Serna**  
Universidad Nacional de Educación a Distancia, Spain

**Juan Manuel Fernández Luna**  
Universidad de Granada, Spain

**Enrique Amigó Cabrera**  
Universidad Nacional de Educación a Distancia, Spain



# Contents

|   |           |
|---|-----------|
| List of figures.....  | v         |
| List of tables .....  | vii       |
| Abstract.....   | xiii      |
| Resumen.....  | xv        |
| Acknowledgements .....  | xvii      |
| <br>  |           |
| <b>Part I. Introduction and context</b>                       | <b>1</b>  |
| <b>Chapter 1. Introduction</b>                                | <b>3</b>  |
| 1.1 Motivation .....  | 4         |
| 1.2 Research goals.....                                       | 6         |
| 1.3 Contributions .....                                       | 7         |
| 1.4 Publications related to the thesis.....                   | 9         |
| 1.5 Structure of the thesis .....                             | 14        |
| <b>Chapter 2. Recommender systems</b>                         | <b>17</b> |
| 2.1 Formulation of the recommendation problem.....            | 18        |
| 2.2 Recommendation techniques.....                            | 19        |
| 2.3 Combining recommender systems.....                        | 28        |
| 2.4 General limitations of recommender systems.....           | 31        |
| 2.5 Summary .....   | 35        |
| <br>  |           |
| <b>Part II. Evaluating performance in recommender systems</b> | <b>37</b> |
| <b>Chapter 3. Evaluation of recommender systems</b>           | <b>39</b> |
| 3.1 Introduction .....  | 40        |
| 3.2 Evaluation metrics.....                                   | 42        |
| 3.3 Experimental setup .....                                  | 49        |
| 3.4 Evaluation datasets.....                                  | 50        |
| 3.5 Summary .....   | 52        |

**Chapter 4. Ranking-based evaluation of recommender systems:  
experimental designs and biases** 53

|     |   |    |
|-----|---|----|
| 4.1 | Introduction.....                           | 54 |
| 4.2 | Cranfield paradigm for recommendation ..... | 55 |
| 4.3 | Experimental design alternatives .....      | 57 |
| 4.4 | Sparsity bias.....                          | 64 |
| 4.5 | Popularity bias .....                       | 67 |
| 4.6 | Overcoming the popularity bias.....         | 70 |
| 4.7 | Conclusions.....                            | 75 |

**Part III. Predicting performance in recommender systems** 77**Chapter 5. Performance prediction in Information Retrieval** 79

|     |   |     |
|-----|---|-----|
| 5.1 | Introduction .....                      | 80  |
| 5.2 | Query performance predictors .....      | 85  |
| 5.3 | Clarity score.....                      | 93  |
| 5.4 | Evaluating performance predictors ..... | 97  |
| 5.5 | Summary.....                            | 102 |

**Chapter 6. Performance prediction in recommender systems** 103

|     |  |     |
|-----|--|-----|
| 6.1 | Research problem.....  | 104 |
| 6.2 | Clarity for preference data: adaptations of query clarity..... | 106 |
| 6.3 | Predictors based on social topology .....                      | 119 |
| 6.4 | Other approaches.....  | 120 |
| 6.5 | Experimental results .....                                     | 123 |
| 6.6 | Conclusions.....   | 136 |

**Part IV. Applications** 139**Chapter 7. Dynamic recommender ensembles** 141

|     |   |     |
|-----|---|-----|
| 7.1 | Problem statement .....   | 142 |
| 7.2 | A performance prediction framework for ensemble recommendation .. | 143 |
| 7.3 | Experimental results .....  | 146 |
| 7.4 | Conclusions.....  | 161 |

---

|   |            |
|---|------------|
| <b>Chapter 8. Neighbour selection and weighting in user-based collaborative filtering</b> | <b>163</b> |
| 8.1 Problem statement .....   | 164        |
| 8.2 A performance prediction framework for neighbour scoring .....                        | 167        |
| 8.3 Neighbour quality metrics and performance predictors .....                            | 171        |
| 8.4 Experimental results.....   | 179        |
| 8.5 Conclusions .....   | 190        |
| <b>Part V. Conclusions</b>  | <b>193</b> |
| <b>Chapter 9. Conclusions and future work</b>   | <b>195</b> |
| 9.1 Summary and discussion of contributions.....  | 196        |
| 9.2 Future work.....  | 198        |
| <b>Part VI. Appendices</b>  | <b>203</b> |
| <b>Appendix A. Materials and methods</b>  | <b>205</b> |
| A.1 Datasets.....   | 206        |
| A.2 Configuration of recommendation algorithms .....                                      | 210        |
| A.3 Configuration of evaluation methodologies .....                                       | 214        |
| A.4 Additional results about correlations.....  | 216        |
| A.5 Additional results about dynamic ensembles .....                                      | 222        |
| <b>Appendix B. Introducción</b>   | <b>229</b> |
| B.1 Motivación.....   | 230        |
| B.2 Objetivos.....  | 232        |
| B.3 Contribuciones.....   | 233        |
| B.4 Publicaciones relacionadas con la tesis.....  | 236        |
| B.5 Estructura de la tesis .....  | 240        |
| <b>Appendix C. Conclusiones y trabajo futuro</b>  | <b>243</b> |
| C.1 Resumen y discusión de las contribuciones.....  | 244        |
| C.2 Trabajo futuro.....   | 246        |
| <b>References</b>   | <b>251</b> |



# List of figures

|  |    |
|--|----|
| Figure 4.1. Precision of different recommendation algorithms on MovieLens 1M and Last.fm using AR and 1R configurations. ....  | 61 |
| Figure 4.2. Empiric illustration of Equation (4.2). The curves show 1RP@10 and its bounds, for pLSA and kNN over MovieLens 1M. The light and dark shades mark the distance to the upper and lower bounds, respectively. The left side shows the evolution when progressively removing test ratings, and the right side displays the variation with $T_u$ ranging from 100 to 1,000. ...  | 63 |
| Figure 4.3. Evolution of the precision of different recommendation algorithms on MovieLens 1M, for different degrees of test sparsity. The x axis of the left and center graphics shows different amounts of removed test ratings. The x axis in the right graphic is the size of the target item sets. ....   | 66 |
| Figure 4.4. Effect of popularity distribution skewness on the popularity bias. The left graphic shows the cumulated popularity distribution of artificial datasets with simulated ratings, with skewness ranging from $\alpha = 0$ to 2. The x axis represents items by popularity rank, and the y axis displays the cumulative ratio of ratings. The central graphic shows the precision of different recommendation algorithms on each of these simulated datasets. The right graphic shows the cumulative distribution of positive ratings in real datasets. .... | 69 |
| Figure 4.5. Rating splits by a) a popularity percentile partition (left), and b) a uniform number of test ratings per item (right). On the left, the red dashed split curve represents $E[pr_{test}(i)]$ – i.e. the random split ratio needs not be applied on a per-item basis – whereas on the right it does represent $pr_{test}(i)$ . ....   | 71 |
| Figure 4.6. Positive ratings ratio vs. popularity rank in MovieLens 1M. The graphic plots $pr(i)/r(i)$ , where items are ordered by decreasing popularity. We display averaged values for 100 popularity segments, for a smoothed trend view.....  | 73 |

|   |     |
|---|-----|
| Figure 4.7. Precision and nDCG of recommendation algorithms on MovieLens 1M<br>(and Last.fm only where indicated) using the 1R, U1R, P1R ( $m = 10$<br>percentiles), AR, and UAR methodologies. The “-10% head” bars show<br>the effect of removing the 10% most popular items from the test data<br>(Cremonesi et al., 2010). ....   | 75  |
| Figure 6.1. Term contributions for each user, ordered by their corresponding<br>contribution to the user language model. IRUserItem clarity model....   | 113 |
| Figure 6.2. User language model sorted by collection probability.....   | 114 |
| Figure 6.3. Heatmap of the correlation values between a subset of predictors and<br>recommenders, using the most representative methodologies for the<br>three considered spaces. ....  | 135 |
| Figure 7.1. Valid predictor correlation regions for a recommender ensemble of size 2.<br>.....  | 145 |
| Figure 7.2. For each best and worst dynamic ensemble in Table 7.2, Table 7.11 and<br>Table 7.15, this graph plots the difference in correlation between each<br>predictor and a recommender against the difference in performance<br>between the ensemble and the baseline.....   | 161 |
| Figure 8.1. Performance comparison for user-based predictors and different<br>neighbourhood sizes. ....   | 184 |
| Figure 8.2. Performance comparison using user-item and user-user predictors for<br>different neighbourhood sizes. ....  | 185 |
| Figure 8.3. Performance comparison using ranking-based metrics for both user and<br>user-user neighbour predictors using the AR and 1R evaluation<br>methodologies. ....  | 187 |
| Figure A.1. Friend distribution among the users composing the original test set of the<br>CAMRa ’10 social track. Note that a logarithmic regression line fits<br>almost perfectly with the above distribution. The total number of users is<br>439, and the maximum and minimum numbers of friends are 14 and 2<br>respectively..... | 209 |

# List of tables

|   |     |
|---|-----|
| Table 2.1. List of common problems in CBF, CF, and SF systems. ....   | 33  |
| Table 4.1. Fitting the recommendation task in the Cranfield IR evaluation paradigm .....  | 56  |
| Table 4.2. Design alternatives in target item set formation.....  | 58  |
| Table 4.3. Notation summary.....  | 59  |
| Table 5.1. Overview of predictors presented in Section 5.2 categorised according to<br>the taxonomy presented in (Carmel and Yom-Tov, 2010). ....   | 84  |
| Table 5.2. Examples of clarity scores for related queries. ....   | 95  |
| Table 5.3. Left: minimum $t$ -value for obtaining a significant value with different<br>sample sizes (N). Right: $t$ -value for a given Pearson's correlation value<br>and N points. In bold when the correlation is significative for $p < 0.05$ ,<br>and underlined for $p < 0.01$ . .... | 100 |
| Table 6.1. Three possible user clarity formulations, depending on the interpretation<br>of the vocabulary and context spaces. ....  | 109 |
| Table 6.2. Different user clarity models implemented.....   | 112 |
| Table 6.3. Two example users, showing the number of ratings they have entered, and<br>their performance prediction values for three user clarity models. ....   | 112 |
| Table 6.4. Three possible item clarity formulations, depending on the interpretation<br>of the vocabulary and context spaces. ....  | 115 |
| Table 6.5. Different item clarity models implemented. ....  | 115 |
| Table 6.6. Two temporal user clarity formulations, depending on the interpretation of<br>the vocabulary space.....  | 118 |
| Table 6.7. Pearson's correlation between rating-based user predictors and P@10 for<br>different recommenders using the AR methodology (MovieLens dataset).<br>.....   | 123 |

|  |     |
|--|-----|
| Table 6.8. Summary of recommender performance using different evaluation methodologies (evaluation metric is P@10 with the MovieLens dataset).   | 124 |
| Table 6.9. Pearson's correlation between rating-based user predictors and P@10 for different recommenders using the 1R methodology (MovieLens dataset).  | 125 |
| Table 6.10. Pearson's correlation between rating-based user predictors and P@10 for different recommenders using the U1R methodology (MovieLens dataset).....  | 126 |
| Table 6.11. Pearson's correlation between rating-based user predictors and P@10 for different recommenders using the P1R methodology (MovieLens dataset).....  | 126 |
| Table 6.12. Procedure to obtain ranking for items from user rankings generated by a standard recommender. * denotes a relevant item, and the numbers are the score predicted by the recommendation method..... | 128 |
| Table 6.13. Pearson's correlation for rating-based item predictors and precision using the uuU1R methodology (MovieLens dataset).....  | 129 |
| Table 6.14. Pearson's correlation between log-based predictors and P@10 for different recommenders using 1R methodology (Last.fm temporal dataset).....  | 130 |
| Table 6.15. Pearson's correlation between log-based predictors and P@10 for different recommenders using 1R methodology (Last.fm five-fold dataset).....   | 131 |
| Table 6.16. Pearson's correlation between social-based predictors and P@10 for different recommenders using AR methodology (CAMRa Social).....   | 132 |
| Table 6.17. Pearson's correlation between social-based predictors and P@10 for different recommenders using AR methodology (CAMRa Collaborative).<br>.....   | 132 |
| Table 7.1. Selected recommenders for building dynamic ensemble using user performance predictors that exploit rating-based information (MovieLens dataset).....  | 148 |

|  |     |
|--|-----|
| Table 7.2. Dynamic ensemble performance values (P@10) using AR methodology and user predictors (MovieLens dataset). Improvements over the baseline are in bold, the best result for each column is underlined. The value $a$ of each dynamic hybrid is marked with $a_y^x$ , where $x$ and $y$ indicate, respectively, statistical difference with respect to the best static (upper, $x$ ) and with respect to the baseline (lower, $y$ ). Moreover, $\blacktriangle$ and $\triangle$ indicate, respectively, significant and non-significant improvements over the corresponding recommender. A similar convention with $\blacktriangledown$ and $\triangledown$ indicates values below the recommender performance. Statistical significance is established by paired Wilcoxon $p < 0.05$ in all cases. ... | 149 |
| Table 7.3. Dynamic ensemble performance values (P@10) using 1R methodology and user predictors (MovieLens dataset).....  | 150 |
| Table 7.4. Dynamic ensemble performance values (P@10) using the U1R methodology and user predictors (MovieLens dataset) .....  | 151 |
| Table 7.5. Dynamic ensemble performance values (P@10) using the P1R methodology and user predictors (MovieLens dataset). ....  | 152 |
| Table 7.6. Selected recommenders for building dynamic ensembles using item predictors that exploit rating data (MovieLens dataset).....  | 153 |
| Table 7.7. Dynamic ensemble performance values (P@10) using 1R methodology with item predictors (MovieLens dataset).....   | 153 |
| Table 7.8. Dynamic ensemble performance values (P@10) using U1R methodology with item predictors (MovieLens dataset).....  | 154 |
| Table 7.9. Dynamic ensemble performance values (P@10) using uuU1R methodology with item predictors (MovieLens dataset).....  | 155 |
| Table 7.10. Selected recommenders for building dynamic ensembles using performance predictors that exploit log-based information (Last.fm dataset).....  | 155 |
| Table 7.11. Dynamic ensemble performance values (P@10) using the 1R methodology with the log-based user predictors (Last.fm, temporal split). .....  | 156 |

|   |     |
|---|-----|
| Table 7.12. Dynamic ensemble performance values (P@10) using the 1R methodology with log-based user predictors (Last.fm, five-fold random split).....   | 156 |
| Table 7.13. Selected recommenders for building dynamic ensembles using social-based user predictors (CAMRa dataset).....  | 157 |
| Table 7.14. Dynamic ensemble performance values (P@10) using the AR methodology with social-based user predictors (CAMRa, social dataset).<br>.....   | 158 |
| Table 7.15. Dynamic ensemble performance values (P@10) using the AR methodology with social-based user predictors (CAMRa, collaborative dataset).....   | 159 |
| Table 8.1. Pearson's correlation between the proposed neighbour quality metrics and neighbour performance predictors in the MovieLens 100K dataset. Next to the metric name, an indication about the sign of the metric – direct(+) or inverse(-) – is included. Not significant values for a <i>p</i> -value of 0.05 are denoted with an asterisk (*). ..... | 180 |
| Table 8.2. Spearman's correlation between quality metrics and performance predictors in the MovieLens 100K dataset.....   | 180 |
| Table 8.3. Pearson's correlation between quality metrics and performance predictors in the MovieLens 1M dataset. All the values are significant for a <i>p</i> -value of 0.05.....  | 181 |
| Table 8.4. Spearman's correlation between quality metrics and predictors in the MovieLens 1M dataset.....   | 181 |
| Table 8.5. Correlation between the user-neighbour goodness and user-user predictors in the two datasets evaluated.....  | 183 |
| Table 8.6. Detail of the accuracy of baseline vs. recommendation using neighbour weighting; here, performance predictors are used as similarity scores (50 neighbours).....   | 186 |
| Table 8.7. Detail of the accuracy of baseline vs recommendation using neighbour selection; here, performance predictors are used for filtering (50 neighbours).....   | 186 |

|   |     |
|---|-----|
| Table A.1. Summary of statistics of the MovieLens 100K dataset .....  | 206 |
| Table A.2. Summary of statistics of the MovieLens 1M dataset .....  | 207 |
| Table A.3. Summary of statistics of the Last.fm datasets.....   | 208 |
| Table A.4. Summary of statistics of the CAMRa datasets.....   | 210 |
| Table A.5. List of the recommenders evaluated in this thesis, and the chapters where<br>their evaluations are reported.....   | 211 |
| Table A.6. Configuration of the evaluation methodologies used in the thesis .....   | 215 |
| Table A.7. Pearson's correlation values between rating-based user predictors and<br>MAP@10 for different recommenders, using the AR methodology on the<br>MovieLens 1M dataset .....  | 217 |
| Table A.8. Pearson's correlation values between rating-based user predictors and<br>recall@10 for different recommenders, on the MovieLens 1M dataset and<br>using the AR methodology.....  | 218 |
| Table A.9. Pearson's correlation values between rating-based predictors and<br>nDCG@50 for different recommenders, on the MovieLens 1M dataset<br>and using the AR methodology.....   | 218 |
| Table A.10. Spearman's correlation values between rating-based user predictors and<br>P@10 for different recommenders, on the MovieLens 1M dataset and<br>using the AR methodology.....   | 219 |
| Table A.11. Kendall's correlation values between rating-based user predictors and<br>P@10 for different recommenders, on the MovieLens 1M dataset and<br>using the AR methodology.....  | 219 |
| Table A.12. Pearson's correlation values between rating-based user predictors and<br>nDCG@50 for different recommenders, on the MovieLens 100K dataset<br>and using the AR methodology. ....  | 219 |
| Table A.13. Pearson's correlation values between rating-based user predictors and<br>P@10 for different recommenders, on the MovieLens 1M dataset and<br>using the AR methodology, but considering all the items in test set as<br>relevant. .... | 220 |
| Table A.14. Pearson's correlation values between log-based user predictors and<br>MAP@10 for different recommenders, on the Last.fm temporal dataset<br>and using the 1R methodology. ....  | 221 |

|   |     |
|---|-----|
| Table A.15. Pearson's correlation values between log-based user predictors and MAP@10 for different recommenders, on the Last.fm five-fold dataset and using the 1R methodology.....      | 221 |
| Table A.16. Pearson's correlation values between social-based user predictors and MAP@10 for different recommenders, on the CAMRa social dataset and using the AR methodology.....        | 222 |
| Table A.17. Pearson's correlation values between social-based user predictors and MAP@10 for different recommenders, on the CAMRa collaborative dataset and using the AR methodology..... | 222 |
| Table A.18. Dynamic ensemble performance values (MAP@10) using the rating-based user predictors, on the MovieLens 1M and using the AR methodology.....                                    | 223 |
| Table A.19. Dynamic ensemble performance values (MAP@10) using the rating-based user predictors, on the MovieLens 1M dataset and using the 1R methodology.....                            | 224 |
| Table A.20. Dynamic ensemble performance values (MAP) using the rating-based item predictors, on the MovieLens 1M dataset and using the uuU1R methodology.....                            | 225 |
| Table A.21. Dynamic ensemble performance values (MAP@10) using the log-based user predictors, on the Last.fm temporal split and using the 1R methodology.....                             | 226 |
| Table A.22. Dynamic ensemble performance values (MAP@10) using the log-based user predictors, on the Last.fm five-fold split and using the 1R methodology.....                            | 226 |
| Table A.23. Dynamic ensemble performance values (MAP@10) using the log-based user predictors, on the CAMRa social dataset and using the AR methodology.....                               | 227 |
| Table A.24. Dynamic ensemble performance values (MAP@10) using the log-based user predictors, on the CAMRa collaborative dataset and using the AR methodology.....                        | 228 |

# Abstract

Personalised recommender systems aim to help users access and retrieve relevant information or items from large collections, by automatically finding and suggesting products or services of likely interest based on observed evidence of the users' preferences. For many reasons, user preferences are difficult to guess, and therefore recommender systems have a considerable variance in their success ratio in estimating the user's tastes and interests. In such a scenario, self-predicting the chances that a recommendation is accurate before actually submitting it to a user becomes an interesting capability from many perspectives. Performance prediction has been studied in the context of search engines in the Information Retrieval field, but there is little if any prior research of this problem in the recommendation domain.

This thesis investigates the definition and formalisation of performance prediction methods for recommender systems. Specifically, we study adaptations of search performance predictors from the Information Retrieval field, and propose new predictors based on theories and models from Information Theory and Social Graph Theory. We show the instantiation of information-theoretical performance prediction methods on both rating and access log data, and the application of social-based predictors to social network structures.

Recommendation performance prediction is a relevant problem per se, because of its potential application to many uses. Thus, we primarily evaluate the quality of the proposed solutions in terms of the correlation between the predicted and the observed performance on test data. This assessment requires a clear recommender evaluation methodology against which the predictions can be contrasted. Given that the evaluation of recommender systems is an open area to a significant extent, the thesis addresses the evaluation methodology as a part of the researched problem. We analyse how the variations in the evaluation procedure may alter the apparent behaviour of performance predictors, and we propose approaches to avoid misleading observations.

In addition to the stand-alone assessment of the proposed predictors, we research the use of the predictive capability in the context of one of its common applications, namely the dynamic adjustment of recommendation methods and components. We research approaches where the combination leans towards the algorithm or the component that is predicted to perform best in each case, aiming to enhance the performance of the resulting dynamic configuration. The thesis reports positive empirical evidence confirming both a significant predictive power for the proposed methods in different experiments, and consistent improvements in the performance of dynamic recommenders employing the proposed predictors.



# Resumen

Los sistemas de recomendación personalizados tienen como objetivo ayudar a los usuarios en el acceso y recuperación de información u objetos relevantes en vastas colecciones mediante la sugerencia automática de productos o servicios de potencial interés, basándose en la evidencia observada de las preferencias de los usuarios. Las preferencias de usuario son difíciles de predecir por muchos motivos y, por tanto, los sistemas de recomendación tienen una variabilidad considerable en su tasa de acierto al intentar estimar los gustos e intereses de cada usuario. En este escenario la autopredicción de las probabilidades de que una recomendación sea acertada antes de proporcionarla al usuario se convierte en una capacidad interesante desde múltiples perspectivas. La predicción de eficacia ha sido estudiada en el contexto de los motores de búsqueda en el campo de la Recuperación de Información, pero apenas se ha investigado en el dominio de la recomendación.

Esta tesis investiga la definición y formalización de métodos de predicción de eficacia para sistemas de recomendación. Concretamente, se estudian adaptaciones de predictores de eficacia de búsqueda en el campo de la Recuperación de Información, y se proponen nuevos predictores basados en modelos y técnicas de la Teoría de la Información y la Teoría de Grafos Sociales. Se propone la instanciación de métodos de teoría de información para predicción de eficacia tanto en datos de valoraciones de usuario explícitas como en registros de accesos, así como la aplicación de predictores sociales sobre estructuras de red social.

La predicción de eficacia de recomendación es un problema relevante por sus múltiples usos y aplicaciones potenciales. Por ello, en primer lugar se evalúa la calidad de las soluciones propuestas en términos de la correlación entre la eficacia estimada y la observada en los datos de test. Esta valoración requiere una metodología clara de evaluación de sistemas de recomendación con la que las predicciones puedan ser contrastadas. Dado que la evaluación de los sistemas de recomendación es aún un área de investigación en buena medida abierta, la tesis aborda la metodología de evaluación como parte del problema a investigar. Se analizan entonces cómo las variaciones en el procedimiento de evaluación pueden alterar la percepción del comportamiento de los predictores de eficacia, y se proponen aproximaciones para evitar observaciones engañosas.

Además de las valoraciones independientes de los predictores propuestos, investigamos el uso de su capacidad predictiva en el contexto de una de sus aplicaciones comunes, a saber, el ajuste dinámico de métodos híbridos para combinar algoritmos y componentes de recomendación. Se investigan aproximaciones donde la combinación se inclina hacia el algoritmo o la componente que se predice va a tener mejor eficacia en cada caso, a fin de mejorar la eficacia de la configuración dinámica resultante. La tesis presenta resultados empíricos positivos que confirman tanto un poder predictivo significativo para los métodos propuestos, como consistentes mejoras en la eficacia de recomendaciones dinámicas que utilizan los predictores propuestos.



# Acknowledgements

This manuscript is the result of several years of work in the college, master, and finally PhD studies. In all this time I have received the support of my family, friends, and colleagues, to whom I am very grateful. These lines are a sign of my gratitude to all of them.

First, I would like to thank my supervisors Pablo Castells and Iván Cantador for their constant encouragement at any aspect I had to face these years, from the stressful talks to the (sometimes confusing) reviews received, and including those about the overwhelming first classes as a teacher and the tiresome paperwork. Thanks to Pablo for giving me the opportunity to pursue this PhD with him, for his effort and continuous work during these years, and for allowing me to participate in conferences and projects where I have learnt so much about deadlines and priorities, and how to deal with them according to a given purpose. Special thanks also to Iván whose continuous support helped me to improve my writing and design skills, although there is still some room for improvement in this respect, especially in the latter. Besides, I started my research training with him, which let me grow professionally, and represented my first satisfactions in the form of papers.

Fernando Díez deserves a special mention, since without his counsel and early interest I probably would not belong to my research group, and I may not have done any research at all. I am also deeply grateful and in debt to all of the current and past members of the IRG/NETS group, especially to David Vallet, Miriam Fernández, Sergio López, Pedro Campos, Saúl Vargas, Nacho Fernández, Víctor Villasante, José Conde, and María Medina for all the great times spent together and the really interesting discussions about research, programming, teaching, and life in general. I would also like to thank the rest of occupants of the B-408 and previously those of B-402 (Alexandra Dumitrescu, Enrique Chavarriaga, César Álvarez, Yolanda Torres, David Alfaya, Víctor López, Alejandro García, Manuel Torres, Laura Hernández, Álvaro Valera, Luis Martín, Chema Martínez, and Sara Schwartzman), and the people from the VPULab/GTI, especially José María Martínez, Jesús Bescós, Víctor Valdés, Javier Molina, Fernando López, Marcos Escudero, and Álvaro García with whom I shared projects and meetings.

After my two internships at the University College London, I am very grateful to Jun Wang for his kind support, and also to Tamas Jambor and Jagadeesh Gorla with whom I had the pleasure of collaborating and discussing whenever I needed. Outside of my group, I also met many interesting people with whom I had the opportunity to

discuss about research, learn many things, and even collaborate together; thus, I want to extend my gratitude to all of them, in particular, my special thanks to Javier Parapar, Miguel Martínez, Denis Parra, Xavier Amatriain, Joaquín Pérez, Neal Lathia, Alan Said, Zeno Gantner, Yue Shi, Marcel Blattner, Anmol Bashin, Gunnar Schröder, Sandra García, Owen Phelan, José Ramón Pérez, Ronald Teijeira, Álvaro Barbero, Miguel Ángel Martínez, Alfonso Romero, Eduardo Graells, Marko Tkalcic, and Arkaitz Zubiaga. I am also in debt with the people behind the Apache Mahout open source project since I have used it almost from the beginning of my training process and it offered me the implementations of my first recommender systems.

Gracias a la gente que conocí en el máster (allá por el 2008-2009), sobre todo con los que colaboré de alguna u otra manera, y con los que compartí charlas y discusiones sobre las asignaturas tanto dentro como fuera de la facultad. Muchas gracias a Richard Rojas, Joaquín Fernández, Rafael Martínez, José Miguel Rojas, Clemente Borges y Sergio García. Y si agradezco a los que me acompañaron en el máster, no voy a ser menos con aquellos con los que hice la carrera, la ‘doble’, los que sobrevivimos a la primera promoción de una carrera donde aprendí mucho, tanto académica, como personalmente. Muchas personas merecen este reconocimiento, pero se lo dedico en especial a Juanda, mi compañero de prácticas durante casi toda la carrera y una de las personas con las que más he aprendido: gracias. También gracias a Roberto, Maite, Jorge, Willy, Irene, Edu, Elena, Jesús, Fran y María por conseguir que esos cinco años pasaran mucho más rápido.

I should also mention the grants that supported my research: the fundings for travel, conference, and publication expenses covered by different European and Spanish research projects (references FP6-027685, TSI-2006-26928-E, CENIT-2007-1012, TIN-2008-06566-C04-02, S2009TIC-1542, CCG10-UAM/TIC-5877, and TIN2011-28538-C02-01), the pre-doctoral grants offered by Universidad Autónoma de Madrid (‘Ayudas para inicio de estudios en programas de posgrado 2007’ and ‘Formación de personal investigador FPI 2009’, which additionally covered the expenses of my second internship at UCL), and the teaching assistant position (‘Profesor Ayudante’) offered in 2010 by the same university, which is currently funding this PhD.

También gracias a mis alumnos, con el de este año ya van siete los grupos a los que he dado clase, y aún no termino de acostumbrarme a la sensación de ver cómo alguien confía plenamente en lo que dices para después observar cómo ha aprendido a hacer cosas que un momento antes no imaginaba que podría hacer. Esa sensación la tendré siempre conmigo. Además, agradezco a los profesores de la Escuela Politécnica Superior y del Departamento de Matemáticas todo lo aprendido entonces como alumno, y ahora como profesor, donde desde el otro lado puedo apreciar aún más lo difícil que es ejercer como docente, y cómo algunos os empeñabais en que pareciera muy sencillo: muchísimas gracias. También merecen un agradecimiento especial el personal de Secretaría y de Administración de la Escuela, a los que he traí-

do de cabeza (sobre todo últimamente) con todas las solicitudes, justificantes y demás papeleo necesario en estos últimos 10 años, gracias sobre todo a Marisa Moreno, María José García, Juana Calle y Amelia Martín. Por supuesto, este agradecimiento también se extiende al personal de biblioteca, conserjería, limpieza y cafetería, que hacen que el tiempo que se pasa en la facultad (a veces más del deseable) sea más sencillo, agradable y ameno.

A mis amigos más cercanos, algunos de ellos del colegio, como Emiliano, Raúl, María, Julito, Luis, Gonzalo, Juan, Esther y Claudio; y otros de después, como Roberto, Noha, Alba y Gema; a todos ellos también les agradezco el haber estado ahí todos estos años y no desesperar cuando no podía salir porque estaba siempre ocupado, y por escuchar y mostrar interés cuando se atrevían a preguntar qué hacía en la universidad. Un abrazo muy grande y un gracias transoceánico a mi buen amigo Hugo, que me enseñó que siempre hay que perseguir lo que uno desea, y con el que siempre podré contar, como cuando hablábamos a deshoras durante mi estancia en Londres o trasnochando para acabar algún artículo.

Por último, y por supuesto, no menos importante, a mi familia. Gracias a mis hermanos por obligarme a jugar a la consola para despejarme, y a hablar de fútbol, baloncesto o música aún cuando llevamos varios días sin vernos por nuestros horarios desacoplados, eso hizo más fácil pasar fuera el tiempo necesario para haber podido terminar esta tesis. Gracias a mi madre, de la que he aprendido a trabajar duro para seguir adelante y que el esfuerzo es lo único que cuenta; ella es y siempre ha sido mi inspiración. Y gracias a Susana y a su familia, por acogerme desde el principio como uno más, y por interesarse siempre por mi tesis. Sin duda, Susana ha sido una de las tres personas que más de cerca ha sufrido todo este proceso, sólo me queda agradecerle el haber estado conmigo todo este tiempo y el haber sido capaz de convencerme de que después de cada rechazo habría algo positivo y que, después de todo, yo sería capaz de superarlo. Gracias, no lo habría conseguido sin ti.

Alejandro Bellogín  
October 2012



A mi madre, mis hermanos y Susana



# Part I

## Introduction and context

*Antes de nada has de saber  
que no soy recomendable.*

Ismael Serrano



# **Chapter 1**

## **Introduction**

In this chapter we present a general overview of the thesis. In Sections 1.1 and 1.2 we provide the motivations and research goals of our work. In Section 1.3 we summarise the main contributions of the thesis, and in Section 1.4 we list the publications resulting from our research. Finally, in Section 1.5 we describe the structure of this document.

## 1.1 Motivation

Information Retrieval (IR) technologies have gained outstanding prevalence in the last two decades with the explosion of massive online information repositories, and much in particular the World Wide Web. IR systems are researched and designed in ways that seek to maximise the degree of satisfaction of certain objective conditions, typically – though not necessarily only – user satisfaction. IR research and development have revolved around the definition of models and algorithms that best achieve this goal, methodologies and metrics that let assess how well the goal is achieved by different systems, and sound theories providing a solid ground and orientation in the development of IR algorithms and their consistent evaluation. Among many new trends stemming from this main stream of research and developments, a new research goal started to be considered by the early 2000's: is it possible to predict how good a result returned by an IR system is going to be, before presenting it to the user, or even, before running the IR system at all (Cronen-Townsend et al., 2002)? This question has given rise to a fertile strand of research on so-called **performance prediction** in IR.

Performance prediction has many potential uses in IR. From the user's perspective it may provide valuable feedback that can be used to direct a search, from the system's perspective it may help to distinguish poorly performing queries, and from the system administrator's perspective it may let identify queries related to a specific subject that are difficult for the search engine. Performance prediction approaches are based on the analysis and characterisation of the evidence used by an IR system to assess the relevance (utility, value, etc.) of retrieval objects (documents, goods, etc.) at execution time (Cronen-Townsend et al., 2002). The most classic and basic retrieval scenario involves a user query and a collection of documents as the basic input to form a ranked list of search results, but other additional elements can be taken into account to select and rank results (Baeza-Yates and Ribeiro-Neto, 2011). Any information the retrieval system takes as input can be taken as input for the performance prediction as well, and often the prediction methods use additional information beyond that. The user context (current tasks, query logs, preferences, etc.), global properties of the document collection, comparisons with respect to other reference elements such as historic data, and the output from other systems, among others, are some examples of the different sources of information that a predictor may draw evidence from.

Predicting the performance of a subsystem, module, function, or input by contrasting the performance estimation for a query for each component, enables an array of dynamic optimisation strategies that select at runtime the option which is predicted to work best or, when larger systems or hybrid approaches are used, allows for adjusting on the fly the participation of each module. The IR field is pervaded with

cases where information relevance, retrieval systems, models, and criteria are based on a fusion or combination of sub-models. Personalised retrieval systems (including techniques such as personalised search, recommender systems, collaborative filtering, and retrieval in context) are clear examples where performance prediction can be applied since such systems combine several sources of evidence for relevance assessment, such as explicit queries, search history, explicit user ratings, social information, user feedback, and context models.

Performance prediction finds additional motivation in personalised recommendation, inasmuch these applications may decide to produce recommendations or hold them back, delivering only the sufficiently reliable ones. Furthermore, current Recommender Systems (RS) are characterised by an increasing diversification of the types and sources of data, content, evidence and methods, available to make decisions and build their output. In such context, predicting the performance of a specific recommendation approach or component becomes an appealing problem, as it lets properly combine the available alternatives, and make the most of them by dynamically adapting the recommendation strategy to the situation at hand. The question gains increasing relevance today, with the proliferation of hybrid recommendation techniques to improve the accuracy of the methods – the Netflix prize was a paradigmatic example of the use of this, where all the top ranked participants used combinations of large sets of recommendation methods. This calls for the research of hybrid approaches with a level of dynamic self-adjustment mechanisms, in order to optimise the resulting effectiveness of the recommendation systems, by opportunistically taking advantage of high-quality data when available, but avoiding sticking to fixed strategies when they can be predicted to yield poor results under certain conditions.

Performance prediction in IR is typically assessed in terms of the correlation between a predictor's scores and a system's performance values on a per-query basis. This requires reliable performance evaluation metrics and methodologies, which have been thoroughly analysed, and are currently well established in the IR field, mostly oriented to ad-hoc search. In contrast, evaluation in the RS field is more open, and the variability in evaluation approaches and experimental configurations is significant. How to measure the performance of a recommender system is a key issue in our research since the system quality measurements may be influenced by statistical properties of the measurement approach and/or the experimental design. Throughout this thesis we shall focus on the accuracy of the system, where we have to avoid that if a metric – i.e., precision – is biased towards some form of noise along with the recommender's quality, then a predictor capturing only that noise would appear as an (equivocal) effective performance predictor. Hence, statistical biases (noises) of the evaluation methodologies should be well understood in order to enable a meaningful assessment of performance predictors.

Drawing from the state of the art on performance prediction in IR as a starting point, the present work restates the problem in the field of Recommender Systems where it has barely been addressed so far. We research meaningful definitions of performance in the context of RS, and the elements to which it can sensibly apply, investigating the statistical biases that may arise when adapting the IR evaluation framework into RS. In doing so, we take as a driving direction the application of performance prediction to achieve improvements in two specific combination problems in the RS field, namely, the dynamic combination of recommendation methods in hybrid recommendation systems, and the dynamic aggregation of neighbours' signals in user-based collaborative filtering.

## 1.2 Research goals

The main objective of the research presented here is to find predictive methods for the performance of specific components in recommender systems, and to improve the performance of combined recommendation methods, based on the dynamic, automatic analysis and prediction of the expected performance of the constituents of the composite methods, whereupon the relative participation of each constituent is adjusted, in accordance to its predicted effectiveness. To address these problems, this work has the following specific research objectives:

**RG1: Analysis and formalisation of how retrieval performance is defined and evaluated in recommender systems.** We need to develop an in-depth study on how recommender systems can be reliably evaluated in terms of numeric metric values, since we aim to predict their performance. Moreover, we have to investigate whether there is any bias on the way the systems are evaluated – either by the evaluation methodologies or metrics, since any bias in the evaluation process would lead to inconclusive or misleading results about the predictive power of the performance prediction methods proposed. If these biases do exist, we aim to precisely understand them and develop methodologies to isolate them; then, we shall check the effectiveness of the predictors against well-known baselines and whether it changes when unbiased methodologies are used.

**RG2: Adaptation and definition of performance prediction techniques for recommender systems.** We aim to study the potential of performance prediction in specific problems and settings in the area of Recommender Systems. We shall investigate the definition of a formal framework where performance predictors can be integrated. As a starting point, we aim to explore the adaptation of specific effective predictors from Information Retrieval such as query clarity (Cronen-Townsend et al., 2002) to recommender systems. Complementarily to the adaptation of known techniques, we aim to research the definition of new predictors based on models from Information Theory and Social Graphs, besides other heuristic, domain-specific ap-

proaches. Once we have defined some recommendation performance predictors, we shall assess the effectiveness of such predictors in terms of their correlation to performance metrics to estimate the predictive power of the performance predictors.

**RG3: Application of performance predictors to hybrid and compound recommender systems.** We aim to identify and integrate the proposed predictors into combined recommendation methods, in order to achieve an actual improvement in the performance of the combined methods. With this goal in mind, we shall consider problems where an aggregation of recommendation methods is needed, and shall analyse how to apply the performance predictors mentioned above in such problems. Besides, a methodological study for the experimental approach, setup, and metrics should be performed in such a way that appropriate baseline methods and experimental designs are used. Finally, we shall assess the improvements and benefits of the combined methods when the performance predictors are applied.

## 1.3 Contributions

This thesis is devoted to the problem of estimating the performance of recommender systems for particular users and items. The main contributions of this thesis are related to the evaluation of the performance of a recommender system, and the prediction of such performance, where we have addressed several issues regarding both topics and we have proposed novel models and methods, which have been applied into two applications as we shall see next.

As a first step, this thesis analyses the Cranfield paradigm of Information Retrieval evaluation since recommender systems are usually considered as a particular problem of information filtering, and, thus, of information retrieval at large (Belkin and Croft, 1992). In Chapter 4 we discuss the differences involved in the experimental design alternatives from the common assumptions made in the Cranfield paradigm, which result in substantial statistical biases arising in Recommender Systems, and we propose different methods to neutralise these biases. Additionally, the following related contributions have been addressed:

- We propose a precise and systematic characterisation of design alternatives in the adaptation of the Cranfield paradigm to recommendation tasks. We identify assumptions and conditions underlying the Cranfield paradigm that are not granted in usual recommendation experiments.
- We detect and characterise resulting statistical biases, namely test sparsity and item popularity, which do not arise in common test collections from IR, but do interfere in recommendation experiments.

- We propose two novel experimental designs in order to neutralise these biases. We observe that a percentile-based evaluation considerably reduces the margin for the popularity bias, whereas a uniform-test approach removes any statistical advantage provided by having more positive test ratings. Furthermore, we find that both approaches discriminate well between pure popularity-based recommendation and an efficient personalised recommendation algorithm.

Additionally, in this thesis **we show how query performance prediction techniques developed in Information Retrieval can be adapted to Recommender Systems, and result in effective predictors in this domain**. We present these performance predictors in Chapter 6, where we propose different adaptations of the query clarity predictor based on different interpretations of the underlying language models along with models from Information Theory and Social Graphs. Furthermore, in the same chapter **we assess the effectiveness of such predictors by measuring the correlation with respect to performance metrics**, where we also test the methods proposed in Chapter 4 to neutralise biases on evaluation. Specific contributions regarding performance prediction for recommendation are summarised as follows:

- We define and elaborate several predictive models in the Recommender Systems domain according to different formulations and assumptions, and based on three types of preference data: rating-based, log-based, and social-based.
- Formulations for rating preferences are based on adaptations of query clarity from IR and concepts from Information Theory such as entropy. In this adaptation we propose different probability estimations, where Bayesian derivations and non-parametric estimations are developed.
- We also exploit temporal features when defining log-based predictors. Specifically, we use a time-aware version of the Kullback-Leibler divergence, along with other time series concepts such as a user's autocorrelation.
- We use graph-based metrics from Graph Theory to define predictors leveraging social network structures, and correlations between topological properties of users and the success of recommendations delivered to them.
- We find strong correlations between the outputs of the predictors and the performance metrics, thus finding empirical evidence of the predictive power of the proposed approaches. Furthermore, when unbiased evaluation methodologies are used, the predictors still obtain good correlation values, evidencing that our proposed predictors are not just capturing and benefitting from the analysed biases, especially when we compare them against other trivial predictors.

Finally, Chapters 7 and 8 present two applications of performance predictors on Recommender Systems. In Chapter 7 **we propose several linearly weighted hy-**

brids where the weights are dynamically adjusted based on the predictors' output. We observe that the correlations obtained in Chapter 6 help decide which are the best combinations to experiment with. More importantly, the correlation between the predictor and the recommender tends to anticipate well when a hybrid will outperform its baseline. Besides, Chapter 8 presents a unified framework where the performance predictors are used to select and weight nearest neighbours in a standard user-based collaborative filtering algorithm. The standard methodology from performance prediction is adapted and translated into this problem, where novel neighbour performance metrics are defined and the predictive power of the predictors is assessed.

The contributions related to the application part of the thesis are, in summary:

- We propose a dynamic hybrid framework to automatically decide when and how dynamic hybridisation should be done, depending on different conditions, namely the correlations between the recommenders and the predictors, and the relative performance level of the combined recommenders.
- In several experiments with the aforementioned performance predictors, our results indicate that a strong correlation with performance tends to correspond with enhancements in dynamic hybrid recommendation when the predictors are used for the adjustment of the combination weights.
- We propose a theoretical framework for neighbour selection and weighting in user-based recommender systems. This framework is based on performance prediction by casting the neighbourhood-based rating prediction task as a case of dynamic output aggregation.
- We compare several state-of-the-art rating-based trust metrics and other proposed neighbour scoring techniques, interpreted as neighbour performance predictors. We also propose several neighbour performance metrics that capture different notions of neighbour quality.

## 1.4 Publications related to the thesis

In the following international journal and conference papers we presented descriptions, results and conclusions related to this thesis:

### Performance prediction and evaluation

1. Bellogín, A., Cantador, I., Díez, F., Castells, P., and Chavarriaga, E. (2012). An empirical comparison of social, collaborative filtering, and hybrid recommenders. *ACM Transactions on Intelligent Systems and Technology*, to appear.

2. Bellogín, A., Castells, P., and Cantador, I. (2011). Predicting the Performance of Recommender Systems: An Information Theoretic Approach. In Amati, G. and Crestani, F., editors, *ICTIR*, volume 6931 of *Lecture Notes in Computer Science*, pages 27–39, Berlin, Heidelberg. Springer Berlin / Heidelberg.
3. Bellogín, A., Castells, P., and Cantador, I. (2011). Self-adjusting hybrid recommenders based on social network analysis. In *Proceedings of the 34<sup>th</sup> international ACM SIGIR conference on Research and development in Information*, SIGIR ’11, pages 1147–1148, New York, NY, USA. ACM.
4. Bellogín, A., Castells, P., and Cantador, I. (2011). Precision-oriented evaluation of recommender systems: an algorithmic comparison. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys ’11, pages 333–336, New York, NY, USA. ACM.
5. Bellogín, A. and Castells, P. (2010). A Performance Prediction Approach to Enhance Collaborative Filtering Performance. In Gurrin, C., He, Y., Kazai, G., Kruschwitz, U., Little, S., Roelleke, T., Rüger, S., and Rijsbergen, editors, *Advances in Information Retrieval*, volume 5993 of *Lecture Notes in Computer Science*, pages 382–393, Berlin, Heidelberg. Springer Berlin / Heidelberg.
6. Bellogín, A. and Castells, P. (2009). Predicting neighbor goodness in collaborative filtering. In And, T. A., Yager and, R. R., And, H. B., And, H. C., and Larsen, H. L., editors, *FQAS*, volume 5822 of *Lecture Notes in Computer Science*, pages 605–616, Berlin, Heidelberg. Springer Berlin / Heidelberg.

## Content-based recommendation

7. Cantador, I., Bellogín, A., and Vallet, D. (2010). Content-based recommendation in social tagging systems. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys ’10, pages 237–240, New York, NY, USA. ACM.
8. Cantador, I., Bellogín, A., and Castells, P. (2008). News@hand: A Semantic Web Approach to Recommending News. In Nejdl, W., Kay, J., Pu, P., and Herder, E., editors, *Adaptive Hypermedia and Adaptive Web-Based Systems*, volume 5149 of *Lecture Notes in Computer Science*, chapter 34, pages 279–283. Springer Berlin / Heidelberg, Berlin, Heidelberg.
9. Cantador, I., Bellogín, A., and Castells, P. (2009). Ontology-Based Personalised and Context-Aware Recommendations of News Items. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-LAT ’08. IEEE/WIC/ACM International Conference on*, volume 1, pages 562–565.
10. Cantador, I., Bellogín, A., Fernández-Tobías, I., and López-Hernández, S. (2011a). Semantic Contextualisation of Social Tag-Based Profiles and Item Recommendations. In Huemer, C., Setzer, T., Aalst, W., Mylopoulos, J.,

- Sadeh, N. M., Shaw, M. J., Szyperski, C., Aalst, W., Mylopoulos, J., Sadeh, N. M., Shaw, M. J., and Szyperski, C., editors, *Electronic Commerce and Web Technologies*, volume 85 of *Lecture Notes in Business Information Processing*, chapter 9, pages 101–113. Springer Berlin Heidelberg, Berlin, Heidelberg.
11. Fernández-Tobías, I., Cantador, I., and Bellogín, A. (2011). cTag: Semantic contextualisation of social tags. In *Proceedings of the Workshop on Semantic Adaptive Social Web (SASWeb 2011)*. CEUR Workshop Proceedings, vol. 730, pages 45–54. RWTH, Aachen (2011).

### **Collaborative filtering recommendation**

12. Bellogín, A., Wang, J., and Castells, P. Bridging Memory-Based Collaborative Filtering and Text Retrieval. *Information Retrieval Journal*, to appear.
13. Bellogín, A., Cantador, I., and Castells, P. A Comparative Study of Heterogeneous Item Recommendations in Social Systems. *Information Sciences*, to appear.
14. Bellogín, A. and Parapar, J. (2012). Using Graph Partitioning Techniques for Neighbour Selection in User-Based Collaborative Filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, RecSys '12, pages 213–216, New York, NY, USA. ACM. (best short paper award)
15. Bellogín, A., Wang, J., and Castells, P. (2011). Structured collaborative filtering. In *Proceedings of the 20<sup>th</sup> ACM international conference on Information and knowledge management*, CIKM '11, pages 2257–2260, New York, NY, USA. ACM.
16. Bellogín, A., Wang, J., and Castells, P. (2011). Text Retrieval Methods for Item Ranking in Collaborative Filtering. In Clough, P., Foley, C., Gurrin, C., Jones, G., Kraaij, W., Lee, H., and Murdoch, V., editors, *Advances in Information Retrieval*, volume 6611 of *Lecture Notes in Computer Science*, chapter 30, pages 301–306. Springer Berlin / Heidelberg, Berlin, Heidelberg.
17. Bellogín, A., Cantador, I., and Castells, P. (2010). A study of heterogeneity in recommendations for a social music service. In *Proceedings of the 1<sup>st</sup> International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, HetRec '10, pages 1–8, New York, NY, USA. ACM.

### **Social filtering recommendation**

18. Díez, F., Chavarriaga, J. E., Campos, P. G., and Bellogín, A. (2010). Movie recommendations based in explicit and implicit features extracted from the Filmtipset dataset. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa '10, pages 45–52, New York, NY, USA. ACM.

### Time-aware recommendation

19. Campos, P. G., Bellogín, A., Díez, F., and Cantador, I. (2012). Time Feature Selection for Identifying Active Household Members. In *Proceedings of the 21<sup>st</sup> ACM international conference on Information and knowledge management*, CIKM '12, New York, NY, USA. ACM (to appear).
20. Campos, P. G., Díez, F., and Bellogín, A. (2011). Temporal rating habits: a valuable tool for rating discrimination. In *Proceedings of the 2<sup>nd</sup> Challenge on Context-Aware Movie Recommendation*, CAMRa '11, pages 29–35, New York, NY, USA. ACM.
21. Campos, P. G., Bellogín, A., Díez, F., and Chavarriaga, J. E. (2010). Simple time-biased KNN-based recommendations. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa '10, pages 20–23, New York, NY, USA. ACM.

### Hybrid recommender systems

22. Cantador, I., Castells, P., and Bellogín, A. (2011). An enhanced semantic layer for hybrid recommender systems. *International Journal on Semantic Web and Information Systems*, 7(1):44–78.
23. Cantador, I., Bellogín, A., and Castells, P. (2008). A multilayer ontology-based hybrid recommendation model. *AI Commun.*, 21(2-3):203–210.
24. Cantador, I., Castells, P., and Bellogín, A. (2007). Modelling Ontology-based Multilayered Communities of Interest for Hybrid Recommendations. In *Workshop on Adaptation and Personalisation in Social Systems: Groups, Teams, Communities, at the 11th International Conference on User Modeling*.

### Recommender evaluation

25. Bellogín, A., Cantador, I., Castells, P., and Ortigosa, A. (2011). Discerning Relevant Model Features in a Content-based Collaborative Recommender System. In Fürnkranz, J. and Hüllermeier, E., editors, *Preference Learning*, chapter 20, pages 429–455. Springer Berlin Heidelberg, Berlin, Heidelberg.
26. Bellogín, A., Cantador, I., Castells, P., and Ortigosa, A. (2008). Discovering Relevant Preferences in a Personalised Recommender System using Machine Learning Techniques. In *Preference Learning Workshop (PL 2008), at the 8<sup>th</sup> European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2008)*, pages 82–96.

These publications are related to the contents of this thesis as follows. In [4] we analyse different evaluation methodologies available in the recommendation literature (Chapters 3 and 4). In [2], [5], and [6] we define the formulations for the concept of user clarity based on ratings (Chapter 6), whereas in [1] and [3] we define the social-

based predictors (again, Chapter 6). Besides, in [1] and [3] we also investigate the use of performance predictors for dynamic hybrid recommendation (Chapter 7). Moreover, in [5] and [6] we address the problem of neighbour weighting based on neighbour performance predictors (Chapter 8).

Additionally, during the course of the thesis, the research presented here has motivated a number of publications that address broader topics in the field, such as content-based recommendation [7-11], collaborative filtering [12-17], social filtering techniques [18], time-aware recommendation [19-21], hybrid recommender systems [22-24], and recommendation evaluation [25, 26]. These publications have resulted in the use and construction of datasets, the development of algorithms and the research and use of some evaluation methodologies and metrics that appear in this thesis.

### **Additional publications**

Preliminary work towards the approaches presented in this thesis was published in my Master's Thesis entitled "Performance prediction in recommender Systems: Application to the dynamic optimisation of aggregative methods" (Bellogín, 2009); specifically, the concept of performance prediction for recommendation is proposed in such work. Apart from that, the motivation, potential impact, and initial main results of our research were published as contributions in two international doctoral symposiums:

- Bellogín, A. (2011). Predicting performance in recommender systems. Doctoral Symposium. In *Proceedings of the fifth ACM conference on Recommender systems*, Rec-Sys '11, pages 371–374, New York, NY, USA. ACM.
- Bellogín, A. (2011). Performance Prediction in Recommender Systems. Doctoral Symposium. In Konstan, J., Conejo, R., Marzo, J., and Oliver, N., editors, *User Modeling, Adaption and Personalization*, volume 6787 of *Lecture Notes in Computer Science*, pages 401–404, Berlin, Heidelberg. Springer Berlin / Heidelberg.

Furthermore, the following submissions are under revision, some of them closely related to the topics of the thesis:

- Bellogín, A., Castells, P., and Cantador, I. Statistical Biases in IR Metrics for Recommender Systems: A Methodological Framework for the Adaptation of the Cranfield Paradigm. Under review.
- Bellogín, A., Castells, P., and Cantador, I. Neighbour Selection and Weighting in User-Based Recommender Systems: A Performance Prediction Approach. Under review.
- Parapar, J., Bellogín, A., Castells, P., and Barreiro, Á. Relevance-Based Language Modelling for Recommender Systems. Under review.

## 1.5 Structure of the thesis

The thesis is divided into six parts. The first part introduces and motivates the problem addressed, along with a survey of the Recommender Systems field, where this thesis is framed. The second part describes the different evaluation techniques used in the recommender systems literature and provides an analysis of the design alternatives and statistical biases that may arise. The third part gives background knowledge and a literature survey on performance prediction, proposes translations of this concept into the recommender system space, and evaluates the predictive power of these approaches. The fourth part provides two applications of the proposed recommender performance predictors. The fifth part concludes and summarises the main contributions of this thesis. Additional information and details are provided in the last part.

In more detail, the contents of this thesis are distributed as follows:

### Part I. Introduction

- **Chapter 1** presents the motivation, research goals, contributions and publications related to the thesis.
- **Chapter 2** provides an overview of the state of the art in recommender systems, considering a classification of the main types of recommendation approaches. We also describe the weaknesses of the different recommendation techniques and present a broader class of hybrid recommenders that aim to overcome these limitations.

### Part II. Evaluating Performance in Recommender Systems

- **Chapter 3** describes the main evaluation metrics and methodologies used in the recommender systems field. The public datasets commonly used in the field are also described.
- **Chapter 4** provides an analysis and formalisation of the different evaluation methodologies reported in the literature. First, we present a systematic characterisation of the experimental design alternatives. Next, we identify and analyse specific statistical biases arising when some methodologies are applied to recommendation, and propose two alternative experimental designs that effectively neutralise such biases to a large extent.

### Part III. Predicting Performance in Recommender Systems

- **Chapter 5** presents the problem of performance prediction in Information Retrieval, surveys the main research works in that area, both in the definition of (query) performance predictors and also in the predictor evaluation in order to infer their predictive power.

- **Chapter 6** states the problem of performance prediction in recommender systems. We define several performance predictors based on three recommendation input spaces where we qualitatively analyse the predictive power of the predictors.

#### Part IV. Applications

- **Chapter 7** proposes a framework where recommender performance predictors are used to build dynamic hybrid recommender systems. We evaluate these recommenders in the three input spaces previously considered for the definition of performance predictors and using different experimental design alternatives where some statistical biases are neutralised.
- **Chapter 8** restates the user-based recommendation problem, providing a generalisation as a performance prediction problem. We investigate how to adopt this generalisation to define a unified framework where we conduct an objective analysis of the effectiveness (predictive power) of neighbour scoring functions.

#### Part V. Conclusions

- **Chapter 9** concludes with a summary of the main contributions of this thesis, and a discussion about future research lines.

#### Part VI. Appendices

- **Appendix A** provides details about the methods proposed in this thesis: configuration of the recommendation algorithms and parameters of the experimental designs used in the evaluation. Detailed statistics about the datasets used in the experiments are provided, complementary to those given in previous chapters.
- **Appendix B** contains the translation into Spanish of Chapter 1.
- **Appendix C** contains the translation into Spanish of Chapter 9.



# Chapter 2

## Recommender systems

The aim of recommender systems is to assist users in finding their way through huge databases and catalogues, by filtering and suggesting relevant items taking into account or inferring the users' preferences (i.e., tastes, interests, or priorities).

Three types of recommender systems are commonly recognised according to how recommendations are made, namely content-based filtering (CBF), collaborative filtering (CF), and social filtering (SF) systems. A CBF system suggests a user items similar to those she preferred or liked in the past, a CF system suggests a user items that people with similar preferences liked in the past, and a SF system suggests items according to the preferences of the user's social contacts in a social network. Each of these types of recommendations has its own strengths and weaknesses. In order to address and compensate particular shortcomings, combinations of different recommendation approaches are usually developed, forming the so called hybrid filtering (HF) systems.

In this chapter we provide an overview of terminology, techniques, and limitations related to the above types of recommender systems. In Section 2.1 we formalise the problem of recommendation, and introduce the different types of recommendation approaches. Next, in Section 2.2 we describe content-based recommendation approaches, rating- and log-based recommendation approaches – as special cases of collaborative filtering –, and social-based recommendation approaches. In Section 2.3 we then explain generic hybrid filtering approaches. Finally, in Section 2.4 we present particular limitations of each type of recommender systems.

## 2.1 Formulation of the recommendation problem

Collaborative filtering can be considered as the first proposed recommendation approach. The term was coined in the mid 90's by Goldberg and colleagues when developing an automatic filtering system for electronic mail (Goldberg et al., 1992), although sometimes the stereotypes defined in (Rich, 1979) have been considered as an earlier reference. Collaborative filtering has been classed as part of the Information Retrieval area by several authors (Belkin and Croft, 1992; Foltz and Dumais, 1992), who have considered recommender systems as a particular case of information filtering. However, only a few recent attempts have been made at bringing recommender systems and information retrieval models together, by establishing equivalences between them (Wang et al., 2008b; Wang et al., 2008a; Bellogín et al., 2011b). Instead, recommender systems have been traditionally investigated from a different perspective, such as preference prediction and Machine Learning (Breese et al., 1998), upon which the main prediction models and evaluation metrics have been developed.

In this context distinct formulations and notations have been proposed. The overview by Adomavicius and Tuzhilin (2005) are among the most cited. In that work the recommendation problem is defined as follows. Let  $\mathcal{U}$  be a set of users, and let  $\mathcal{I}$  be a set of items. Let  $g: \mathcal{U} \times \mathcal{I} \rightarrow \mathbb{R}$ , where  $\mathbb{R}$  is a totally ordered set, be a utility function such that  $g(u, i)$  measures the gain of usefulness of item  $i$  for user  $u$ . Then, for each user  $u \in \mathcal{U}$ , we aim to choose items  $i^{\max, u} \in \mathcal{I}$ , unknown to the user, which maximise the utility function  $g$ , that is:

$$\forall u \in \mathcal{U}, i^{\max, u} = \arg \max_{i \in \mathcal{I}} g(u, i)$$

Depending on the exploited source of user preference information, and the way in which the utility function  $g$  is estimated for different users, the following two main types of recommender systems are commonly distinguished: 1) content-based recommender systems, in which a user is suggested items similar to those she liked or preferred in the past, and 2) collaborative filtering systems, in which a user is suggested items that people with similar preferences liked in the past. We extend this classification by also considering social recommender systems, i.e., systems in which a user is suggested items that friends – e.g. in an online social network – liked in the past. These systems are related but significantly different from collaborative filtering systems. Moreover, we distinguish two types of collaborative filtering systems, based on the form of their input: systems that exploit explicit user ratings (rating-based systems), and systems that exploit implicit user preference information (log-based systems). The rating assigned to an item by a particular user is typically interpreted as the true utility of that item for the user. There are systems, however, where no explicit ratings are available, but where user interests can be inferred from implicit

feedback information. In order to provide item recommendations in such systems, two plausible approaches do exist: 1) directly exploiting implicit preference data (Wang et al., 2008b; Deshpande and Karypis, 2004; Das et al., 2007; Hu et al., 2008; Linden et al., 2003), and 2) transforming implicit preference data into explicit ratings to be exploited by standard CF strategies (Celma, 2010; Jawaheer et al., 2010; Adams, 2007).

Other types of recommender systems have been considered in the literature, although they will not be described in detail herein; these are knowledge-based, utility-based, and demographic-based recommender systems. They use, respectively, semantic descriptions of the user preferences and item characteristics, an utility function over the items that describes the users' preferences, and demographic information about the users. For further descriptions and examples of these techniques, see (Burke, 2002) and (Ricci et al., 2011).

For any of the above mentioned types of recommender systems, models can be combined to improve their separate performance, or other characteristic of interest, such as the capability of providing more diverse and novel recommendations, and offering better explanations of recommendations. When such a combination is performed, the recommendation approach is considered a hybrid recommender (or hybrid filtering) system (Burke, 2002).

## 2.2 Recommendation techniques

As mentioned above, the main goal of a recommender system is to provide users with the most useful items according to their preferences. For such purpose, different strategies may be used, which can be categorised based on the type of data exploited, namely content-based, rating- and log-based collaborative filtering, and social recommendation strategies. In this section we formalise these strategies. We shall use the following notation. Letters  $u$  and  $v$  will be reserved for users ( $u, v \in \mathcal{U}$ ), whereas  $i$  and  $j$  will denote items ( $i, j \in \mathcal{I}$ ),  $\mathcal{U}$  and  $\mathcal{I}$  being, respectively, the set of users and items in the system. Besides,  $r \in R$  will denote a particular rating value, and  $R$  will be the set of possible rating values, either discrete (typically,  $R = \{1,2,3,4,5\}$ ) or continuous (e.g.  $R = [0,5]$ ). Finally,  $\tilde{r}$  shall denote a rating prediction (as opposed to observed ratings denoted by  $r$ ).

### 2.2.1 Content-based recommenders

Content-based filtering (CBF, or simply content-based) techniques recommend items similar to those previously liked by a user. An extensive survey of this type of techniques can be found in (Lops et al., 2011; Pazzani and Billsus, 2007), and

(Adomavicius and Tuzhilin, 2005). In this section we briefly discuss some of the main approaches proposed in the field.

Content-based recommendation algorithms build a user's profile based on the features of the objects rated by the user, which are assumed to reflect the user's content-based interests (Lops et al., 2011). In general, a CBF technique can be classified according to whether a model is built from underlying data, commonly based on Machine Learning techniques (Lops et al., 2011; Pazzani and Billsus, 1997; de Gemmis et al., 2008), or use a heuristic function to compute item scores, mainly inspired on Information Retrieval methods (Diederich and Iofciu, 2006; Balabanovic and Shoham, 1997; Cantador et al., 2010).

Probabilistic methods in general and the naïve Bayes approach in particular generate a probabilistic model based on previously observed data. The naïve Bayes model estimates the *a posteriori* probability  $P(c|d)$  of document  $d$  belonging to class  $c$ , based on the *a priori* probability  $P(c)$  for the class, the probability  $P(d)$  of observing the document, and the probability  $P(d|c)$  of observing the document given the class (Lops et al., 2011), as follows:

$$P(c|d) = \frac{P(c)P(d|c)}{P(d)}$$

In recommendation the naïve Bayes method is used to estimate the probability that a document (an item) is either relevant or irrelevant (class  $c$ ), based on the information available for each user, that is, documents already rated are used to build the  $P(d|c)$  probabilities. This approach has been used by many different authors (Mooney and Roy, 2000; Semeraro et al., 2007; de Gemmis et al., 2008; Lops et al., 2011).

Alternative methods for classifying the items in a system as relevant or irrelevant for each user include decision trees and neural networks (Pazzani and Billsus, 1997). These techniques, similarly to the naïve Bayes method, estimate in which class each (unrated or unobserved) item fits best with the user's profile.

Techniques based on Information Retrieval methods are specified by the way users and items are represented and the similarity function used between them. Typically they use a vector space model where each feature is weighted in a particular way. For instance, instead of using the frequency of each feature in a user/item profile, more complex functions from the Information Retrieval field may be used, such as TF-IDF and BM25 (Cantador et al., 2010). Furthermore, many different feature spaces have been considered in the literature: keywords (Lieberman, 1995; Pazzani et al., 1996), tags (Diederich and Iofciu, 2006; Michlmayr and Cazer, 2007), and semantic concepts enriched by different techniques (Magnini and Strapparava, 2001; Eirinaki et al., 2003; Cantador, 2008).

Regarding the feature vector similarity, the most common measure is the cosine similarity, even though the standard dot product between two vectors has also been used (Cantador et al., 2010):

$$sim_{dot}(d_i, d_j) = \sum_k w_{ki} w_{kj} \quad (2.1)$$

$$sim_{cos}(d_i, d_j) = \frac{sim_{dot}(d_i, d_j)}{\sqrt{\sum_k w_{ki}^2} \sqrt{\sum_k w_{kj}^2}} \quad (2.2)$$

where  $w_{ki}$  is the weight assigned (by any of the techniques mentioned before) to the feature  $k$  in document  $i$ .

In recommender systems items are suggested by decreasing order of similarity with the user, whose profile is represented in the same form of the documents (that is, in the space of features under consideration). The similarities are computed as the feature vector similarity between each (unrated or unobserved) document in the collection and the user's vector.

## 2.2.2 Rating-based recommenders

Collaborative filtering (CF) techniques match people with similar preferences, or items with similar choice patterns from users, in order to make recommendations. Unlike CBF, CF methods aim to predict the utility of items for a particular user according to the items previously evaluated by other like minded users. These methods have the interesting property that no item descriptions are needed to provide recommendations, since the methods merely exploit information about past ratings. Compared to CBF approaches, CF also has the salient advantage that a user may benefit from other people's experience, thereby being exposed to potentially novel recommendations beyond her own experience (Adomavicius and Tuzhilin, 2005).

In this section we focus on those CF techniques based on explicit numeric ratings, which are the most common in the literature. For additional references, see (Desrosiers and Karypis, 2011; Koren and Bell, 2011) and (Adomavicius and Tuzhilin, 2005). Most of our discussion nonetheless applies to log-based recommenders alike. In fact, as we shall show in the next section, most of the rating-based techniques can be used when no ratings are available (although the equivalence introduces additional assumptions).

In general, CF approaches are commonly classified into two main categories: **model-based** and **memory-based**. Model-based approaches build statistical models of user/item rating patterns to provide automatic rating predictions. Some approaches learn such models by performing some form of dimensionality reduction in order to uncover latent factors between users and items, e.g. by such techniques as

Singular Value Decomposition (SVD) for matrix factorisation (Billsus and Pazzani, 1998; Koren et al., 2009), probabilistic Latent Semantic Analysis (pLSA), or Latent Dirichlet Allocation (LDA) (Hofmann, 2003; Blei et al., 2003). Other approaches use probabilistic models where the recommendation task is modelled by user and item probability distributions (Wang et al., 2006b; Wang et al., 2008a), e.g. by learning a probabilistic model with a maximum entropy estimation (Pavlov et al., 2004; Zitnick and Kanade, 2004), Bayesian networks (Breese et al., 1998), and Boltzmann machines (Salakhutdinov et al., 2007). A graph-based model that exploits positive and negative preference data is proposed in (Clements et al., 2009). Besides, other Machine Learning techniques have also been proposed, such as artificial neural networks (Billsus and Pazzani, 1998) and clustering strategies (Kohrs and Merialdo, 1999; Cantador and Castells, 2006).

Memory-based approaches, on the other hand, make rating predictions based on the entire rating collection (Adomavicius and Tuzhilin, 2005; Desrosiers and Karypis, 2011). These approaches can be user- and item-based strategies. **User-based** strategies are built on the principle that a particular user's rating records are not equally useful to all other users as input for providing personal item suggestions (Herlocker et al., 2002). Central aspects to these algorithms are thus a) how to identify which neighbours form the best basis to generate item recommendations for the target user, and b) how to properly make use of the information provided by them. Typically, neighbourhood identification is based on selecting those users who are more similar to the target user according to a similarity metric (Desrosiers and Karypis, 2011). The similarity between two users is generally computed by a) finding a set of items that both users have interacted with, and b) examining to what degree the users displayed similar behaviors (e.g. rating, browsing and purchasing patterns) on these items. This basic approach can be complemented with alternative comparisons of virtually any user feature a system has access to, such as personal demographic and social network data. It is also common practice to set a maximum number of neighbours (or a minimum similarity threshold) to restrict the neighbourhood size either for computational efficiency, or in order to avoid noisy users who are not similar enough. Once the target user's neighbours are selected, the more similar a neighbour is to the user, the more her preferences are taken into account as input to produce recommendations. For instance, a common user-based approach consists of predicting the relevance of an item for the target user by a linear combination of her neighbours' ratings, weighted by the similarity between the target user and such neighbours.

In the following equations we present two versions of a user-based CF technique; in the first one rating deviations from the user's and neighbour's rating means are considered (Resnick et al., 1994), whereas in the second one the raw scores given by each neighbour are used (Aggarwal et al., 1999; Shardanand and Maes, 1995):

$$\tilde{r}(u, i) = \bar{r}(u) + C \sum_{v \in N_k(u, i)} sim(u, v)(r(v, i) - \bar{r}(v)) \quad (2.3)$$

$$\tilde{r}(u, i) = C \sum_{v \in N_k(u, i)} sim(u, v)r(v, i) \quad (2.4)$$

where  $C$  is a normalisation factor (different in each formulation) and  $N_k(u, i)$  is a neighbourhood of size  $k$ , which may use information of the target item  $i$ . As stated in (Adomavicius and Tuzhilin, 2005), these techniques simply use a different function to aggregate the ratings from the neighbourhood. Note that in this case the utility function  $g(u, i)$  is assumed to be equivalent to the predicted rating  $\tilde{r}(u, i)$ , although alternative transformations could be applied if required. Additionally, the similarity between two users is generally computed by means of the Pearson's correlation coefficient or the cosine similarity between the vectors representing each user's preferences (Adomavicius and Tuzhilin, 2005):

$$sim_{Pearson}(u, v) = \frac{\sum_i (r(u, i) - \bar{r}(u))(r(v, i) - \bar{r}(v))}{\sqrt{\sum_i (r(u, i) - \bar{r}(u))^2} \sqrt{\sum_i (r(v, i) - \bar{r}(v))^2}} \quad (2.5)$$

$$sim_{Cosine}(u, v) = \frac{\sum_i r(u, i)r(v, i)}{\sqrt{\sum_i r(u, i)^2} \sqrt{\sum_i r(v, i)^2}} \quad (2.6)$$

Note that these similarities are equivalent when the data is centered on the mean. Nonetheless, some authors have reported that the performance of recommenders based on Pearson's similarity is superior to that of cosine's (Breese et al., 1998; Herlocker et al., 1999). Moreover, other similarity measures and modifications on how the neighbours are selected and weighted have been proposed, either by modifying the similarity measure (McLaughlin and Herlocker, 2004; Ma et al., 2007), by using clustering methods to compute a user's neighbourhood (O'Connor and Herlocker, 1999; Xue et al., 2005), or by learning the best interpolation weights for rating prediction (Bell and Koren, 2007; Koren, 2008). Additionally, in the context of trust-based recommendation, the neighbours are weighted (and selected) according to their importance from the target user's point of view (O'Donovan and Smyth, 2005; Weng et al., 2006; Kwon et al., 2009; Hwang and Chen, 2007).

**Item-based** strategies, on the other hand, recognise patterns of similarity between the items themselves, instead of between user choices like user-based approaches do. In general item-based recommenders look at each item on the target user's list of chosen/rated items, and find other items that seem to be "similar" to that item (Shardanand and Maes, 1995; Sarwar et al., 2001). The item similarity is usually defined in terms of rating correlations between users, although cosine-based or probability-based similarities have also been proposed (Deshpande and Karypis,

2004). As stated in (Sarwar et al., 2001), adjusted cosine similarity has been proved to obtain better performance than other item similarities. This similarity subtracts the user's average rating from each co-rated pair in the standard cosine formulation:

$$\text{sim}(i,j) = \frac{\sum_u (r(u,i) - \bar{r}(u))(r(u,j) - \bar{r}(u))}{\sqrt{\sum_u (r(u,i) - \bar{r}(u))^2} \sqrt{\sum_u (r(u,j) - \bar{r}(u))^2}} \quad (2.7)$$

The rating prediction computed by item-based strategies is generally estimated as follows (Sarwar et al., 2001):

$$\tilde{r}(u,i) = C \sum_{j \in S_i} \text{sim}(i,j)r(u,j) \quad (2.8)$$

We have to note that the set of more similar items  $S_i$  is generally replaced by  $\mathcal{I}_u$  – the set of items rated by user  $u$  – since for any other item, the rating provided by the user is assumed to be zero, and thus, it does not contribute to the summation.

### 2.2.3 Log-based recommenders

Different methods have been proposed to use implicit evidence of user preferences. The work of Oard and Kim (1998) represents one of the first attempts to exploit implicit user feedback to estimate future ratings in a recommender system. In general most of recent approaches have used formal models (generally probabilistic) in order to introduce implicit data for recommendation, although some approaches using ad-hoc techniques can be found. For example, Linden et al. (2003) use a simple vector representation, where each component represents purchased items, and recommendations are obtained by ranking each item according to how many similar users purchased it. Bernhardsson (2009) proposes a graph item-based algorithm that finds the closest tracks for a given track using probabilistic LSA (pLSA), and then derives the recommendations using heuristic and model-based probabilities, by brute force.

Additionally, several formal algorithms have been proposed to use implicit user feedback from log data: namely matrix factorisation, such as SVD (Hu et al., 2008) and pLSA (Das et al., 2007), and language models and other probabilistic approaches (Wang et al., 2006a; Wang, 2009; Wang et al., 2008b; Deshpande and Karypis, 2004). These algorithms aim to capture the user's preferences by considering the consumed (purchased, listened, browsed, etc.) items as evidence of positive relevance for the user. This fact often leads to binary models in which the number of times the user has consumed each item is not taken into consideration. Nevertheless, a benefit of using binary data is that it allows to better account for the fact that ratings are not missing at random – or equivalently, that users choose deliberately which items to rate (Marlin et al., 2007). Besides this a general concern about negative preferences has arisen. For instance, in (Lee and Brusilovsky, 2009), (Wang et al., 2008c), and

(Xin and Steck, 2011) the authors attempt to incorporate negative preferences inferred from implicit data.

Other authors have proposed different transformations in order to obtain explicit ratings from implicit feedback. The most naive approach is to make a correspondence between the existence of an item in a log record and a (frequency-independent) rating. For example, the algorithm proposed in (Ali and van Stam, 2004) cannot distinguish an explicit +1 rating from the rating inferred from implicit data. This is the same procedure that can be found in (Lee et al., 2008), but with other transformations based on the time in which an item was entered in the system and consumed.

In (Baltrunas and Amatriain, 2009), based on (Celma, 2010) and (Celma, 2008), the authors use a more elaborate mapping where the number of times a user listened to an artist (or track) is taken into account, in such a way that the artists (tracks) located in the 80-100% interquintile range of the user's playcount distribution receive a rating of value 5 (in a five point scale), the next interquintile range is mapped to a rating of value 4, and so on. This technique has also been used in other works, such as (Vargas and Castells, 2011). A similar technique is presented in (Jawaheer et al., 2010), where three methods are proposed in order to calculate the preference of a user for an artist: i) absolute, where the raw count of the number of times that artist has been played is used; ii) normalised, where the preference is inferred by the ratio between the counts for an artist and the total number of artists played by the user; and iii) logarithmic, similar to the previous one but smoothing the preference values by applying a logarithmic transformation.

Finally, Adams (2007) proposes a complete ad-hoc formula that takes several parameters into consideration, such as the number of times the current track has been played and skipped, the number of seconds when it was skipped, and the number of days since it was last played.

In conclusion, there is no definitive unique method for transforming implicit into explicit data. Moreover, it is unclear to what extent the mapping is reliable (Hu et al., 2008), since it inherently represents different information gathered from the user – for instance, negative preferences can only be fetched using explicit data. However, a recent study reported a strong relation between the amount of times users listen to an album, and the rating they provide to the album (Parra and Amatriain, 2011).

## 2.2.4 Social-based recommenders

Recommender systems that exploit social information, such as contacts and interactions between users, have started to be developed in recent years. We shall henceforth refer to this type of recommendation approaches as Social Filtering (SF) systems. Recommendations by SF approaches have the interesting property that they

are generally easier to explain than user-based CF approaches. Recommendations through friends are indeed easy to interpret by end-users. They also help dealing with the cold start problem, where new users are more difficult to provide recommendations for as long as it is not possible to reliably compute their similarity with other users for lack of data (Golbeck, 2006; Arazy et al., 2009).

Shepitsen et al. (2008) propose a personalisation approach for recommendation in folksonomies that relies on hierarchical tag clusters. The approach suggests the most similar items to the user's closest cluster by means of the cosine similarity measure. Other approaches focus on graph based techniques for finding the most relevant items for a particular user through hybrid networks involving people, items, and tags (Konstas et al., 2009; Clements et al., 2010). In this context alternative methods have been proposed to deal with data sparsity. Besides, prediction accuracy is improved by means of factor analysis based on probabilistic matrix factorisation, employing both the users' social network information and rating records (Ma et al., 2008). Ma et al. (2009) combine the recommendations made by trusted friends with those generated by a matrix factorisation algorithm. In a similar way, Jamali and Ester (2009) propose to perform a random walk on the trust network, considering the similarity of users in the termination condition; then, the top rated items are recommended. Both approaches are competitive in cold start situations.

Complementarily, simpler algorithms (referred to as “pure” social recommenders henceforth) have also been proposed in (Liu and Lee, 2010) and (Bellogín et al., 2012). In (Liu and Lee, 2010) an adaptation of the user-based CF technique is proposed, where the set of nearest neighbours is replaced by the target user's (explicit) friends. That is:

$$N_k(u, i) = \{v \in \mathcal{U} : v \text{ is friend of } u\} \quad (2.9)$$

This lets easily incorporate social information into the CF prediction equation, building a straightforward technique that enables a direct interpretation of the suggestions, namely those items recommended by friends. Similarly, based on a recommender proposed in (Barman and Dabeer, 2010), where the items suggested to a user are the most popular among her set of similar users, in (Bellogín et al., 2012) we proposed a friends' popularity recommender that suggests the target user those items most popular for her set of friends. A score is generated by transforming the item position with the following equation, once a ranking has been generated using the score  $f(u, i)$ :

$$\begin{aligned} f(u, i) &= |\{v \in \mathcal{U} : v \text{ is friend of } u \text{ and } rat(v, i) \neq \phi\}| \\ g(u, i; N) &= 1 - \frac{pos(u, i)}{N} \end{aligned} \quad (2.10)$$

where  $pos(u, i)$  represents the position of item  $i$  in the top- $N$  recommended list for user  $u$ . We may trim the returned list at some level  $N$ , or assume  $N$  to be exactly the

length of the generated recommendation list. Obviously, the computed scores cannot be interpreted as ratings, but as a utility or ranking score. In (Bourke et al., 2011) Bourke and colleagues also make use of the social graph of a user to build the neighbourhood, analysing the perceived trust, which is found to be higher when the users are given the opportunity to manually select the neighbourhood to be used for computing recommendations.

Ben-Shimon et al. (2007) propose a recommendation approach based on the distances between users in the social graph. The approach uses Breadth-First Search to build a social tree for each user  $u$  denoted as  $X(u, L)$ , where  $L$  is the maximum number of levels taken into consideration in the algorithm, and  $K$  is an attenuation coefficient of the social network that determines the extent of the effect of  $d(u, v)$ , that is, the impact of the distance between two users in the social graph (e.g. by using an algorithm that computes the distance between two nodes in a graph, such as the Dijkstra's algorithm (Dijkstra, 1959)). Hence, when  $K = 1$  the impact is constant, and the resulting ranking is sorted by the popularity of the items. Furthermore, for that value of  $K$ , no expansion is applied and only directly connected users are involved in the score computation. Once the value of  $K$  is chosen, a rating score is generated according to the following equation:

$$g(u, i) = \sum_{v \in X(u, L)} K^{-d(u, v)} r(v, i) \quad (2.11)$$

An alternative way of introducing social information into a recommender system is by the so called trust-based recommendation approaches, even though social relationships and trust relationships do not model exactly the same concept (Ma et al., 2011). Trust-aware recommenders, in contrast with those defined in Section 2.2.2, make use of trust networks, where users express a level of trust on other users (Massa and Avesani, 2007a). These recommenders need a trust network and a trust metric, so that trustworthiness of every user can be computed. Depending on the available data, we would have to infer a plausible trust network, from the information we already know about users, such as social interactions among users or explicit trust relations. Typically, uniform trust values from each user are assumed, since no distinction can be made among a user's contacts. For example, a user with 4 friends would have a trust level of 0.25 for each friend, whereas a user with 2 friends would have such trust level of 0.5.

Once the trust network is defined, either explicitly or implicitly, we can set different definitions for the trust metrics depending on whether they are global (a global reputation value is calculated for each user) or local (a trust score is computed between a source user on a target user). Social-based trust metrics make use of explicit trust networks of users, built upon friendship relationships (Massa and Bhattacharjee, 2004) and explicit trust scores between individuals in a system (Ma et al., 2009; Wal-

ter et al., 2009). These metrics and, to some extent, their inherent meanings, are different with respect to rating-based metrics. Nonetheless, Ziegler and Lausen (2004) conduct a thorough analysis that shows empirical correlations between trust and user similarity, suggesting that users tend to create social connections with people who have similar preferences. Once such a correlation is proved, techniques based on social-based trust are applicable.

Golbeck and Hendler (2006) propose a metric called TidalTrust to infer trust relationships by recursive search. Inferred trust values are used for every user who has rated a particular item in order to select only those users with high trust values. Then, a weighted average between ratings and trust provides the predicted ratings. A similar algorithm is used in (Walter et al., 2009), where the prediction is based on the ratings of the trusted neighbours. Different integrations of the trust metric into the recommendation process are proposed in (Massa and Avesani, 2007a), along with two metrics: PageRank and MoleTrust. The former is considered as a global metric based on the well-known PageRank algorithm (Brin and Page, 1998); the latter is a local metric based on a Depth-First graph traversal algorithm with an adjustable trust propagation horizon (Massa and Avesani, 2007a).

Finally, as proposed in (Massa and Avesani, 2007a), two ways to incorporate these trust metrics into the recommendation models can be considered. The first one makes use of the trust metric instead of the similarity metric in the standard user-based CF formula. The second one, on the other hand, computes the average between Pearson's similarity and the trust metric when both values are available; otherwise it uses the only available value, thus overcoming the natural data sparsity. Recently, Guo et al. (2012) propose to merge the ratings from the trusted neighbours in order to decrease sparsity prior to the computation of the predicted rating.

## 2.3 Combining recommender systems

The proliferation of new recommendation strategies is giving rise to an increasing variety of available options for the development of recommender systems. Research in Machine Learning has long shown that the combination of methods usually achieves better results than each method separately, which is also true in Recommender Systems – the Netflix prize has been a paradigmatic example of this, where all the top classified teams used large recommender ensembles, which can be considered as a case of hybrid filtering approaches.

In such a hybrid approach the most important decision is how to combine the information. First, however, it has to be decided what kind of information is going to be used in the ensemble. The standard approach in the literature is to combine CBF and CF recommenders, overcoming the sparsity and restricted feature problems of individual recommenders, as we shall see in the next section. However, other types

and sources of information, such as social contacts and timestamps, have been recently integrated into the classical formulation of standard recommendation techniques.

In (Burke, 2002) a detailed taxonomy of hybrid recommender systems is presented, classifying existing approaches into the following types:

- **Cascade:** the recommendation is performed as a sequential process in such a way that one recommender refines the recommendations given by the other.
- **Feature augmentation:** the output from one recommender is used as an additional input feature for other recommender.
- **Feature combination:** the features used by different recommenders are integrated and combined into a single data source, which is exploited by a single recommender.
- **Meta-level:** the model generated by one of the recommenders is used as the input for other recommender. As stated in (Burke, 2002): “this differs from feature augmentation: in an augmentation hybrid, we use a learned model to generate features for input to a second algorithm; in a meta-level hybrid, the entire model becomes the input.”
- **Mixed:** recommendations from several recommenders are available, and are presented together at the same time by means of certain ranking or combination strategy.
- **Weighted:** the scores provided by the recommenders are aggregated using a linear combination or a voting scheme.
- **Switching:** a special case of the previous type considering binary weights, in such a way that one recommender is turned on and the others are turned off.

The use of a specific type of hybrid recommendation method depends on the final application, but, more importantly, on the type of recommenders being combined. Indeed, Burke (2002) presents an analysis of the possible hybrids, their limits and incompatibilities, based on a representative subset of the recommendation techniques available nowadays. Moreover, the author notes that some combinations turn out to be redundant because of the symmetry in the hybridisation process for some of the techniques listed above: weighted, mixed, switching, and feature combination. Incompatible combinations arise for the feature combination and meta-level techniques, where in some situations one of the recommenders is not able to use the model or the features generated by the other recommender.

Burke (2002) focuses on hybrid techniques where the information being combined consists of ratings (to be used by CF recommenders), content features (to be used by CBF, knowledge-based, and utility-based recommenders), and demographic

information. In the following, we survey hybrid recommenders where inputs in the form of social information, collaborative (either ratings or logs), and content features have been used. In the next section we analyse the limitations of these types of techniques, together with the benefits that hybridisation may bring.

Among these possibilities, the most popular combination (probably due to its inherent interest) consists of blending content-based and collaborative filtering recommenders. In fact, one of the first proposed hybrid techniques (Balabanovic and Shoham, 1997) makes use of these two recommendation approaches by suggesting items similar to the user's profile (using content-based profiles) and those items highly rated by a user with a similar profile, by means of a collaborative formulation where neighbours are determined using a content-based similarity. In a similar way, Pazzani (1999) combines content-based, demographic, and collaborative information using two techniques: by plugging content-based similarity functions into collaborative methods and by combining the final rankings produced by each recommender seeking a consensus, that is, how many systems recommend each item, and in what ranking position are both considered to build the final ranking.

In (Rojasattarat and Soonthornphisaj, 2003) a technique to derive a less sparse pseudo rating matrix is proposed. More specifically, a pseudo user-ratings vector for every user is built with the item ratings provided by user  $u$  when available, or the ratings predicted by a content-based recommender otherwise. Gunawardana and Meek (2009) propose to combine content and collaborative information in a coherent manner by using a specific type of probabilistic models, Boltzmann machines. These models let encoding the above sources of information as features, and then, weights are learned to reflect how each feature helps predict the user ratings. Other probabilistic models for combining these sources of information have been proposed in (Yu et al., 2003), where a hierarchical Bayesian model learns a prior distribution by using probabilistic Support Vector Machines (SVMs).

Also from a machine learning perspective, an ensemble technique known as stacking is used in (Bao et al., 2009), which learns multiple classifiers for different prediction levels: at the first level, the recommendation techniques (a user-based CF, an item-based CF, and a CBF algorithm) output a rating prediction, which may be combined at the second level by a meta-learning algorithm that uses the predictions as meta-features.

Alternatively, the same model can also be combined with itself using different parameter values. For instance, in (Gantner et al., 2010) different factor models are combined, where each model may have different regularisation parameters, stop conditions and dimensionality values. Jahrer et al. (2010) combine a set of diverse CF recommenders by using different machine learning techniques such as linear regression, neural networks, and a combination of bagging and gradient boosting trained with decision trees.

Furthermore, hybrid models have been proposed combining social and content or collaborative information. In (Konstas et al., 2009) a Random Walk algorithm is applied to a graph comprising of tags, social information, and implicit feedback from users. In this way, more elaborate patterns and rules than the standard correlation measure between users are provided. A similar approach can be found in (Liu et al., 2010) for tag recommendation. The approach defined in (Clements et al., 2010) improves search and recommendation by combining tags and ratings, and integrating them into the user's social network also using a Random Walk algorithm. Hotho et al. (2006) exploit social information along with tag content by converting a folksonomy into a graph and then applying a weight-spreading algorithm for folksonomies called FolkRank (similar to the well-known PageRank algorithm (Brin and Page, 1998). Finally, Jamali and Ester (2009) combine information from the social network (in terms of trust between users) and ratings (collaborative) in order to alleviate the cold-start problem. In that work, the authors make use of the collaborative information as a termination condition of a random walk performed over the trust network by considering the similarity of users; additionally, the authors also combine those two sources for computing two sets of neighbours and, then, merging the items produced from those similar users.

## 2.4 General limitations of recommender systems

Each type of recommendation technique has strengths and weaknesses, well known in the field. We have already noted the main characteristics of each technique, which are largely dependent on the source of information being used. In this section we analyse the main limitations of each technique. Furthermore, although ideally hybrid recommendation techniques would overcome the problems of the combined techniques, there are certain limitations that are inherent to the recommendation problem, and thus, have to be addressed independently. Besides, by combining different methods, additional problems, along with more limitations, arise.

### 2.4.1 Limitations of single recommendation algorithms

In this section we describe the different limitations identified in the literature for the main types of recommenders described in the previous sections.

The main limitations of CBF approaches are the following (Adomavicius and Tuzhilin, 2005; Pazzani and Billsus, 2007; Cantador, 2008):

- **Restricted content analysis.** Content-based recommendations depend on the available features explicitly associated with the items. These features should be in a form that can be automatically parsed by a computer, or manually ex-

tracted somehow, which, depending on the domain, could be unfeasible or very difficult to maintain.

- **New user.** A user has to show some preference (ratings) for a sufficient number of items before a recommender can build a reliable content-based user profile.
- **Overspecialisation.** Since content-based recommenders only retrieve items similar to what the user has already rated, recommendations are very similar and, probably, well known to the user, providing little (or none) novelty from the user perspective.
- **Portfolio effect.** Related to the previous limitation, sometimes the recommended items are very similar among them, leading to a set of insufficiently diverse or too redundant item suggestions.

CF approaches have the following general weaknesses:

- **Rating data sparsity.** The number of observed user-item interactions (e.g. ratings) is generally very small compared to the number of all user-item pairs. This fact may cause CF algorithms to produce unreliable recommendations, since they have been inferred from insufficient data.
- **Grey sheep.** Since collaborative recommendations rely on the tastes of similar people to suggest new items, when a user has very specific or unusual preferences, it will be more difficult for the system to find good neighbours, and thus, to recommend interesting items.
- **New item.** Until a new item has been rated by a substantial number of users, a recommender system may not be able to recommend it; hence, popular items tend to have advantage in this kind of systems.
- **New user.** Like in the content-based approaches, until a user has not provided with enough ratings, the system is unable to recommend her interesting unknown items.

In addition to these weaknesses, log-based CF techniques have other limitations. Specifically, they are not able to capture negative preferences from the user since unobserved items cannot be inferred as unliked items (they may represent items unknown for the user). In contrast, it is easier to capture this type of information because it is less expensive for the user than providing a rating. Furthermore, although the problem of ratings missing not at random is ubiquitous and inherent to any recommender system – since users typically rate only a small fraction of the available items – log-based recommenders, and more specifically, the binary data inferred from these implicit interactions, have the theoretical advantage that they are able to exploit implicit preferences since the items observed by the users are deliberately

| Problem                     | Description  | CBF | CF  | SF  |
|-----------------------------|--|-----|-----|-----|
| Restricted content analysis | Items to be recommended must have available data related to their features. This data is often unavailable or incomplete.                          | Yes | No  | No  |
| Overspecialisation          | CBF recommenders are trained with the content features of the items. All the recommended items are similar to those already rated.                 | Yes | No  | No  |
| Portfolio effect            | CBF recommenders suggest items based on the item features. An item is recommended even if it is too similar to a previously rated item.            | Yes | No  | No  |
| New user                    | A user has to rate enough items in order to infer their preferences. When a new user enters into the system she has no ratings.                    | Yes | Yes | No  |
| New item                    | Items have to be rated by a substantial number of users for being recommended. Recently incorporated items have none or insufficient ratings.      | No  | Yes | No  |
| Grey sheep                  | A user has to be similar to others in the community to receive recommendations. Users whose tastes are unusual may not receive useful suggestions. | No  | Yes | No  |
| Rating data sparsity        | Ratings are used to train user and item models. The number of available ratings is usually small.  | No  | Yes | No  |
| Social sparsity             | Social connections are used to build social models. The number of connections per user may be small.   | No  | No  | Yes |
| New social connection       | A user has to be connected with someone else to receive recommendations. When the user is new, she may not have any social connections.            | No  | No  | Yes |
| Social similarity           | Similarity based on social connections is used in SF recommenders. Two users socially connected may or may not have interests in common.           | No  | No  | Yes |

**Table 2.1. List of common problems in CBF, CF, and SF systems.**

selected by them. Thus, potentially more useful information about the user can be gathered (Koren and Bell, 2011).

Regarding CF in general, memory-based approaches achieve lower performance than model-based approaches. However, as stated in (Desrosiers and Karypis, 2011) and (Koren and Bell, 2011), good prediction accuracy does not guarantee an effective and satisfying user experience. Hence, the main advantages of memory-based recommenders are simplicity, justifiability, efficiency, and stability.

Finally, SF approaches have other limitations, as we describe next:

- **Social sparsity.** Social filtering methods need that every user has to be connected through at least one contact in the social network to be able to produce recommendations, which is not a typical situation for most of the users in a system.

- **New social connection.** Recommendations may get biased if a user has a very small social network, up to the point that if she has only one connection, every social recommendation would be generated based on the activity of just one user.
- **Social similarity.** The fact that two users share some kind of connection in a social network does not necessarily mean that these users have similar interests. Although some studies have shown some correlation between both (Ziegler and Lausen, 2004), the misuse of this similarity may lead to bad recommendations, even though the user's experience may be improved in terms of diversity and serendipity.

As a summary, Table 2.1 shows a comparison of the main limitations for the three types of recommendation algorithms described.

#### 2.4.2 Limitations of recommender ensembles

As we have explained in the previous section, each type of recommendation – CBF, CF, and SF – has its own limitations. Hybrid filtering systems are normally out of this analysis since they compensate the shortcomings of one approach by the strengths of the other, unless both suffer from the same problem, as in the case of a new user when we combine CBF and CF approaches.

In general, hybrid recommenders are useful for alleviating the individual limitations of the combined recommenders. However, recommender ensembles do not always outperform individual recommenders. Van Setten (2005) describes the situation where all recommenders produce predictions that are “on the same side of the rating the user would give, all too low or all too high.” In this situation the ensemble would be less accurate than the best individual recommender. Additionally, when a particular recommender always obtains superior/inferior performance than the rest of recommenders in the ensemble, the corresponding recommender ensemble may not be useful. In that case the underperforming recommenders are useless from the beginning, whereas the over performing one should be used alone, and there is no point in combining them.

The above issues assume that a particular metric is aimed to be optimised. Needless to say that the use of multiple recommenders may provide better results with respect to other evaluation properties, such as diversity, novelty, and serendipity, probably at the expense of a lower quality or accuracy of the recommendations (Shani and Gunawardana, 2011).

Additionally, the recommender ensemble problem is similar to that of combining classifiers in the Machine Learning field, a well studied research problem in that community (Kuncheva, 2004). In such context, the diversity in the classifier outputs is known to be a requirement for the combination to be effective. Thus, whenever

some classifiers in an ensemble fail, these errors should be made on different objects, in order to let a final performance improvement with the ensemble. In (Kuncheva, 2004) and (Kuncheva and Whitaker, 2003) Kuncheva and Whitaker present a number of diversity metrics, and analyse the relation of such metrics with respect to the accuracy of a recommender ensemble, although they do not provide a systematically formulation of such relation. As stated by the authors, the problem of classifier combination and its relation with diversity may rise from the underlying meaning of diversity: whether it is a characteristic of the set of classifiers, or it is more complex and a mixture of the characteristics of the set of classifiers, the combiner, and the errors.

Finally, although many different hybrid filtering approaches have been proposed for recommender systems, there is a lack of a similar analysis to the one performed in Machine Learning, where the different characteristics of the datasets and individual recommenders have been investigated and assessed. A preliminary analysis was performed in (Bellogín et al., 2010), but an in-depth and larger-scale study would benefit the community, considering different evaluation perspectives and, probably, borrowing from the Machine Learning research on this topic.

## 2.5 Summary

Along over two decades of research and commercial development, recommender systems have proved to be a successful technology to overcome the information overload that burdens users in modern online media. The inherent possibility of dealing with diverse sources of information, such as the content of the items and the collaborative and social interactions among users and between users and a system, has enabled the development of rich strategies based on each of these evidences, deriving content-based, collaborative, and social filtering recommendation approaches. Furthermore, as each particular type of recommendation technique has its own limitations and weaknesses, hybrid strategies have been proposed that combine the suggestions generated by different techniques in different ways. The success of ensemble approaches has been recently evidenced in the Netflix prize, where the top classified teams used different forms of recommender ensembles.

There are, however, general limitations remain unsolved, and are still considered as open research problems in the field. We have mentioned the sparsity of the information (either in the forms of content-based attributes, collaborative ratings, and social connections), and the new user problem, but other problems, not related to a specific recommendation technique, have been identified in the literature, and deserve special attention by themselves, such as the need of contextualisation, the explanation of the recommendations, and the efficiency in computing recommendations (Adomavicius and Tuzhilin, 2005).



## Part II

# Evaluating performance in recommender systems

*If you cannot measure it, you cannot improve it.*  
William Thomson (Lord Kelvin)



# Chapter 3

## Evaluation of recommender systems

The evaluation of recommender systems has been, and still is, the object of active research in the field. Since the advent of the first recommender systems, recommendation performance has been usually equated to the accuracy of rating prediction, that is, estimated ratings are compared against actual ratings, and differences between them are computed by means of the *mean absolute error* and *root mean squared error* metrics. In terms of the effective utility of recommendations for users, there is however an increasing realisation that the quality (precision) of a ranking of recommended items can be more important than the accuracy in predicting specific rating values. As a result, precision-oriented metrics are being increasingly considered in the field, and a large amount of recent work has focused on evaluating top-N ranked recommendation lists with the above type of metrics.

In this chapter we provide a survey of different evaluation metrics, protocols, and methodologies in the recommender systems field. In Section 3.1 we provide a preliminary overview of how recommender systems are evaluated, presenting the main (online and offline) evaluation protocols and dataset partitioning methods. Next, in Section 3.2 we present the most common evaluation metrics, classified into error-based and precision-based metrics, and in Section 3.3 we describe different dataset partition strategies used in the experimental configurations. Finally, in Section 3.4 we present some evaluation datasets which are commonly used by the research community, and that were used in the experimental work of this thesis.

### 3.1 Introduction

The evaluation of recommender systems has been a major object of study in the field since its earliest days, and is still a topic of ongoing research, where open questions remain (Herlocker et al., 2004; Shani and Gunawardana, 2011). Two main evaluation protocols are usually considered (Gunawardana and Shani, 2009): *online* and *offline*. In this thesis we focus on offline evaluation, which lets compare a wide range of candidate algorithms at a low cost (Shani and Gunawardana, 2011). For a review of the different tasks and protocols for online recommendation evaluation, see (Shani and Gunawardana, 2011), (Pu et al., 2012), and (Kohavi et al., 2009).

Drawing from methodological approaches common to the evaluation of classification, machine learning and information retrieval algorithms, offline recommender system evaluation is based on holding out from the system a part of the available knowledge of user likes (test data), leaving the rest (training data) as input to the algorithm, and requiring the system to predict such preferences, so that the goodness of recommendations is assessed in terms of how the system's predictions compare to the withheld known preferences. In the dominant practice, this comparison has been oriented to measure the accuracy of rating prediction, computing error-based metrics. However, in terms of the effective utility of recommendations for users, there is an increasing realisation that the quality (precision) of the ranking of recommended items can be more important than the accuracy (error) in predicting specific rating values. As stated in (Herlocker et al., 2004), the research community has moved from the *annotation in context* task (i.e., predicting ratings) to the *find good items* task (i.e., providing users with a ranked list of recommended items), which better corresponds to realistic settings in working applications where recommender systems are deployed. As a result, precision-oriented metrics are being increasingly considered in the field. Yet there is considerable divergence in the way such metrics are applied by different authors, as a consequence of which the results reported in different studies are difficult to put in context and be compared.

In the classical formulation of the recommendation problem, user preferences for items are represented as numeric ratings, and the goal of a recommendation algorithm consists of predicting unknown ratings based on known ratings and, in some cases, additional information about users, items, and the context. In this scenario, the accuracy of recommendations has been commonly evaluated by measuring the error between predicted and known ratings, using metrics such as the Mean Absolute Error (MAE), and the Root Mean Squared Error (RMSE). Although dominant in the literature, some authors have argued this evaluation methodology is detrimental to the field since the recommendations obtained in this way are not the most useful for users (McNee et al., 2006). Acknowledging this, recent work has evaluated top-N ranked recommendation lists with precision-based metrics (Cremonesi et al., 2010;

McLaughlin and Herlocker, 2004; Jambor and Wang, 2010b; Bellogín et al., 2011b), drawing from evaluation well studied methodologies in the Information Retrieval field.

Precision-oriented metrics measure the amount of relevant and non-relevant retrieved (recommended) items. A solid body of metrics, methodologies, and datasets has been developed over the years in the Information Retrieval field. Recommendation can be naturally stated as an information retrieval task: users have an implicit need with regards to a space of items which may serve the user's purpose, and the task of the recommender system is to select, rank and present the user a set of items that may best satisfy her need. The need of the user and the qualities or reasons why an item satisfies it cannot be observed in full, or described in an exact and complete way, which is the defining characteristic of an information retrieval problem, as opposed to data retrieval tasks or logical proof. It is thus natural to adapt relevance-based Information Retrieval evaluation methodologies here, which mainly consist of obtaining manual relevance labels of recommended items with respect to the user's need, and assessing, in different ways, the amount of relevant recommended items.

Recommendation tasks and the available data for their evaluation, nonetheless, have specific characteristics, which introduce particularities with respect to mainstream experience in the Information Retrieval field. In common information retrieval experimental practice, driven to a significant extent by the TREC campaigns (Voorhees and Harman, 2005), relevance knowledge is typically assumed to be (not far from) complete – mainly because in the presence of a search query, relevance is simplified to be a user-independent property. However, in recommender systems it is impractical to gather complete preference information for *each* user in a system. In datasets containing thousands of users and items, only a fraction of the items that users like is generally known. The unknown rest are, for evaluation purposes, assumed to be non-relevant. This is a source of – potentially strong – bias in the measurements depending on how unknown relevance is handled. In the next chapter we cover in detail these problems, along with an analysis of the different experimental design alternatives available in the literature.

In the reminder of this chapter we present some of the most common evaluation metrics. We classify them into error-based and precision-based metrics, accounting for the two tasks previously described – rating prediction and item ranking, respectively. After that, we describe the main methodologies used in the area to partition datasets and to select the candidate items in the latter task. Finally, we introduce the datasets used in this thesis to evaluate different recommendation algorithms.

## 3.2 Evaluation metrics

The evaluation of recommender systems should take into account the goal of the system itself (Herlocker et al., 2004). For example, in (Herlocker et al., 2004) the authors identify two main user tasks: *annotation in context* and *find good items*. In these tasks the users only care about errors in the item rank order provided by the system, not the predicted rating value itself. Based on this consideration, researchers have started to use precision-based metrics to evaluate recommendations, although most works also still report error-based metrics for comparison with state of the art approaches. Moreover, other authors, such as Herlocker and colleagues (Herlocker et al., 2004), encourage considering alternative performance criteria, like the novelty of the suggested items and the item coverage of a recommendation method. We describe the above types of evaluation metrics in the subsequent sections.

### 3.2.1 Error-based metrics

A classic assumption in the recommender systems literature is that a system that provides more accurate predictions will be preferred by the user (Shani and Gunawardana, 2011). Although this has been further studied and refuted by several authors (McNee et al., 2006; Cremonesi et al., 2011; Bollen et al., 2010), the issue is still worth being analysed.

Traditionally, the most popular metrics to measure the accuracy of a recommender system have been the **Mean Absolute Error** (MAE), and the **Root Mean Squared Error** (RMSE):

$$\text{MAE} = \frac{1}{|\text{Te}|} \sum_{(u,i) \in \text{Te}} |\tilde{r}(u,i) - r(u,i)| \quad (3.1)$$

$$\text{RMSE} = \sqrt{\frac{1}{|\text{Te}|} \sum_{(u,i) \in \text{Te}} (\tilde{r}(u,i) - r(u,i))^2} \quad (3.2)$$

where  $\tilde{r}$  and  $r$  denote the predicted and real rating, respectively, and  $\text{Te}$  corresponds to the test set. The RMSE metric is usually preferred to MAE because it penalises larger errors.

Different variations of these metrics have been proposed in the literature. Some authors **normalise MAE** and **RMSE** with respect to the maximum range of the ratings (Goldberg et al., 2001; Shani and Gunawardana, 2011) or with respect to the expected value if ratings are distributed uniformly (Marlin, 2003; Rennie and Srebro, 2005). Alternatively, **per-user** and **per-item average errors** have also been proposed in order to avoid biases from the error (or accuracy) on a few very frequent users or items (Massa and Avesani, 2007a; Shani and Gunawardana, 2011). For instance, the user-average MAE is computed as follows:

$$\text{uMAE} = \frac{1}{|U|} \sum_{u \in U} \frac{1}{|\text{Te}_u|} \sum_{i \in \text{Te}_u} |\tilde{r}(u, i) - r(u, i)| \quad (3.3)$$

A critical limitation of these metrics is that they do not make any distinction between the errors made on the top items predicted by a system, and the errors made for the rest of the items. Furthermore, they can only be applied when the recommender predicts a score in the allowed range of rating values. Because of that, log-based, and some content-based and probabilistic recommenders cannot be evaluated in this way, since  $\tilde{r}(u, i)$  would represent a probability or, in general, a preference score. Hence, these methods can only be evaluated by measuring the performance of the generated ranking using precision-based metrics.

### 3.2.2 Precision-based metrics

These metrics can be classified into three groups: metrics that only use one ranking, metrics that compare two rankings (typically, one of them is a reference or ideal ranking), and metrics from the Machine Learning field.

#### Metrics based on one ranking

Examples of these metrics are precision, recall, normalised discounted cumulative gain, mean average precision, and mean reciprocal rank. Each of these metrics captures the quality of a ranking from a slightly different angle. More specifically, **precision** accounts for the fraction of recommended items that are relevant, whereas **recall** is the fraction of the relevant items that has been recommended. Both metrics are inversely related, since an improvement in recall typically produces a decrease in precision. They are typically computed up to a ranking position or cutoff  $k$ , being denoted as  $P@k$  and  $R@k$ , and defined as follows (Baeza-Yates and Ribeiro-Neto, 2011):

$$P@k = \frac{1}{|U|} \sum_{u \in U} \frac{|\text{Rel}_u @ k|}{k} \quad (3.4)$$

$$R@k = \frac{1}{|U|} \sum_{u \in U} \frac{|\text{Rel}_u @ k|}{|\text{Rel}_u|} \quad (3.5)$$

where  $\text{Rel}_u$  represents the set of relevant items for user  $u$ , and  $\text{Rel}_u @ k$  is the number of relevant recommended items up to position  $k$ .

Recall has also been referred to as **hit-rate** in (Deshpande and Karypis, 2004). Hit-rate has also been defined as the percentage of users with at least one correct recommendation (Bellogín et al., 2012), corresponding to the **success** metric (or **first relevant score**), as defined by TREC (Tomlinson, 2005).

Furthermore, the **mean average precision** (MAP) metric provides a single summary of the user's ranking by averaging the precision figures obtained after each new relevant item is obtained, as follows (Baeza-Yates and Ribeiro-Neto, 2011):

$$\text{MAP} = \frac{1}{|\mathcal{U}|} \sum_u \frac{1}{|\text{Rel}_u|} \sum_{i \in \text{Rel}_u} \text{P}@\text{rank}(u, i) \quad (3.6)$$

where  $\text{rank}(u, i)$  outputs the ranking position of item  $i$  in the user's  $u$  list; hence, precision is computed at the position where each relevant item has been recommended.

**Normalised discounted cumulative gain** (nDCG) uses graded relevance that is accumulated starting at the top of the ranking and may be reduced, or discounted, at lower ranks (Järvelin and Kekäläinen, 2002):

$$\text{nDCG} = \frac{1}{|\mathcal{U}|} \sum_u \frac{1}{\text{IDCG}_u^{p_u}} \sum_{p=1}^{p_u} f_{\text{dis}}(\text{rel}(u, i_p), p) \quad (3.7)$$

where the discount function  $f_{\text{dis}}(\text{rel}(u, i_p), p)$  is usually defined as  $f_{\text{dis}}(x, y) = (2^x - 1)/\log(1 + y)$  or simply  $f_{\text{dis}}(x, y) = x/\log y$  if  $y > 1$ ,  $f_{\text{dis}}(x, y) = x$  otherwise, depending on the emphasis required on retrieving highly relevant items (Croft et al., 2009).  $\text{IDCG}_u^k$  denotes the score obtained by an ideal or perfect ranking for user  $u$  up to position  $k$ , which acts as a normalisation factor in order to compare different users and datasets. Besides,  $p_u$  denotes the maximum number of items evaluated for each user; which is typically assumed to be a cutoff  $k$ , the same for all the users. In that situation, this metric is denoted as nDCG@ $k$ .

Using a different discount function, the **rank score** or **half-life utility** metric (Breese et al., 1998; Herlocker et al., 2004; Huang et al., 2006) can be obtained as follows:

$$\text{HL} = 100 \left( \sum_u \text{HL}_u^{\max} \right)^{-1} \sum_u \text{HL}_u; \quad \text{HL}_u = \sum_{p=1}^{p_u} \frac{\max(\tilde{r}(u, i_p) - d, 0)}{2^{(p-1)/(\alpha-1)}} \quad (3.8)$$

where  $d$  is the default ranking, and  $\alpha$  is the half-life utility that represents the rank of the item on the list such that there is a 50% chance that the user will view that item. In (Breese et al., 1998) the authors use a value of 5 in their experiments, and note that they did not obtain different results with a half-life of 10.

**Mean reciprocal rank** (MRR) favours rankings whose first correct result occurs near the top ranking results (Baeza-Yates and Ribeiro-Neto, 2011). It is defined as follows:

$$\text{MRR} = \sum_u \frac{1}{s_r(u)} \quad (3.9)$$

where  $s_r(u)$  is a function that returns the position of the first relevant item obtained for user  $u$ . This metric is similar to the **average rank of correct recommendation** (ARC) proposed in (Burke, 2004) and to the **average reciprocal hit-rank** (ARHR) defined in (Deshpande and Karypis, 2004).

It is important to note that since its early days, there has been a concern in the Information Retrieval field for the value and validity of the standard precision and recall metrics in interactive contexts (Su, 1992; Belkin and Croft, 1992). Nonetheless, precision-based metrics such as precision and recall, and more in general, metrics that measure the quality of the item ranking returned by a recommender have been frequently used in the field, despite they often lead to uncomparable results (Bellogín et al., 2011a).

### Metrics based on two rankings

Additionally, specific metrics have been defined in the context of recommender evaluation that take as inputs two rankings (ideal vs estimated) instead of just one. A first example is the **normalised distance-based performance measure** (NDPM), used in (Balabanovic and Shoham, 1997), and proposed in (Yao, 1995). This metric compares two different weakly ordered rankings, and is formulated as follows (Herlocker et al., 2004; Shani and Gunawardana, 2011):

$$\text{NDPM} = \frac{1}{|\mathcal{U}|} \sum_u \frac{2C_u^{\text{con}} + C_u^{\text{tie}}}{2C_u} \quad (3.10)$$

where  $C_u$  is the number of pairs of items for which the real ranking (reference ranking using the ground truth) asserts an ordering, i.e., the items are not tied. Besides,  $C_u^{\text{con}}$  denotes the number of discordant item pairs between the method's ranking and the reference ranking, and  $C_u^{\text{tie}}$  represents the number of pairs where the reference ranking does not tie, but where the method's ranking does. This metric is comparable across datasets since it is normalised with respect to the worst possible scenario (denominator). Furthermore, it provides a perfect score of 0 to systems that correctly predict every preference relation asserted by the reference, and a worst score of 1 to methods that contradict every reference preference relation. Besides, a penalisation of 0.5 is applied when a reference preference relation is not predicted, whereas predicting unknown preferences (i.e., they are not ordered in the reference ranking) receives no penalisation.

As the previous metric, rank correlation metrics such as **Spearman's  $\rho$**  and **Kendall's  $\tau$**  have also been proposed to directly compare the system ranking to a preference order given by the user. These correlation coefficients are later defined and analysed (Chapter 5). Here we only indicate that they provide scores in the range

of  $-1$  to  $1$ , where  $1$  denotes a perfect correlation between the two above rankings, and  $-1$  represents an inverse correlation.

These two metrics, along with NDPM, suffer from the interchange weakness (Herlocker et al., 2004), that is, interchanges at the top of the ranking have the same weight that interchanges at the bottom of the ranking.

### **Metrics from Machine Learning**

Finally, some other metrics from the Machine Learning literature have also been used, although they are not very popular. For instance, the **receiving operating characteristic** (ROC) curve and the **area under the curve** (AUC) have been used in (Herlocker et al., 1999), (Schein et al., 2001), (Schein et al., 2002), and (Rojasattarat and Soonthornphisaj, 2003), among others. Metrics based on the ROC curve provide a theoretically grounded alternative to precision and recall (Herlocker et al., 2004). The ROC model attempts to measure the extent to which an information filtering system can successfully distinguish between signal (relevant items) and noise. Starting from the origin of coordinates at  $(0,0)$ , the ROC curve is built by considering, at each rank position, whether the item is relevant or not for the user; in the first case, the curve goes one step up, and in the second, one step right.

A random recommender is expected to produce a straight line from the origin to the upper right corner; on the other hand, the more leftwards the curve leans, the better is the performance of the system. These facts are related to the area under the ROC curve, a summary metric that is expected to be higher when the recommender performs better, where the expected value of a random recommender is  $0.5$ , corresponding to a diagonal curve in the unit square.

In (Schein et al., 2001) the authors discriminate between the Global ROC (GROC) curve and the Customer ROC (CROC) curve, where the former assumes that only the most certain recommendations are made where some users may receive no recommendation at all; thus, the number of recommendations could be different for each user. The CROC curve is more realistic in the sense that every user receives the same amount of recommended items. However, for this curve a perfect recommender would not necessarily obtain an AUC of  $1$ , and thus, it is required to compute the associated value of a perfect ROC curve in order to provide a fair comparison and normalise accordingly.

#### **3.2.3 Other metrics**

As different applications have different needs, additional characteristics of recommendations could be taken into consideration, and thus alternative metrics beyond accuracy and precision may be measured. In this context, it is important to understand and evaluate the possible trade-offs between these additional characteristics

and their effect on the overall recommendation performance (Shani and Gunawardana, 2011). For instance, some algorithms may provide recommendations with high quality or accuracy, but only for a small proportion of users or items, probably due to data sparsity. This effect can be quantified by measuring the **coverage** of the recommender system. Two types of coverage can be defined: *user coverage* (proportion of users to whom the system can recommend items) and *item or catalog coverage* (proportion of items the system can recommend). In (Shani and Gunawardana, 2011) two metrics are proposed for measuring item coverage: one based on the Gini's index, and another based on Shannon's entropy. In (Ge et al., 2010) the authors propose simple ratio quantities to measure such metrics, and to discriminate between the percentage of the items for which the system is able to generate a recommendation (*prediction coverage*), and the percentage of the available items that are effectively ever recommended (*catalog coverage*). A similar distinction is considered in (Herlocker et al., 2004) and (Salter and Antonopoulos, 2006). In (Herlocker et al., 2004) it is acknowledged that item coverage is particularly important for the tasks of *find all good items* and *annotation in context*. Besides, a system with low coverage is expected to be less valuable to users and the authors propose to combine coverage with accuracy measures to yield an overall “practical accuracy” measure for the system, in such a way that coverage is raised only because recommenders produce bogus predictions.

Beyond coverage, two recommendation characteristics have become very popular recently: **novelty** and **diversity**. Already a large amount of work has focused on defining metrics for measuring such characteristics (Lathia et al., 2010; Shani and Gunawardana, 2011; Vargas and Castells, 2011; Zhang and Hurley, 2009), and designing algorithms to provide novel and/or diverse recommendations (Jambor and Wang, 2010b; Onuma et al., 2009; Weng et al., 2007; Zhou et al., 2010).

Novel recommendations are those that suggest the user items she did not know about prior to the recommendation (Shani and Gunawardana, 2011), referred to as non-obvious items in (Herlocker et al., 2004; Zhang et al., 2002). Novelty can be directly measured in online experiments by directly asking users whether they are familiar with the recommended item (Celma and Herrera, 2008). However, it is also interesting to measure novelty in an offline experiment, so as not to restrict its evaluation to costly and hardly reproducible online experiments.

Novelty can be introduced into recommendations by using a topic taxonomy (Weng et al., 2007), where items containing novel topics are appreciated. Typically, novel topics are obtained by clustering the previously observed topics for each user. In (Lathia et al., 2010), novelty measures the amount of new items appearing in the recommended lists over time. In (Onuma et al., 2009) a technique based on graphs is introduced to suggest nodes (items) well connected to older choices, but at the same time well connected to unrelated choices.

Metrics based on Information Theoretic properties of the items being recommended have also been proposed by several authors. In (Bellogín et al., 2010) the entropy function is used to capture the novelty of a recommendation list, in (Zhou et al., 2010) the authors use the self-information of the user's top recommended items, and in (Filippone and Sanguinetti, 2010) the Kullback-Leibler divergence is used.

In Information Retrieval, diversity is seen as an issue of avoiding redundancy and finding results that cover different aspects of an information need (Radlinski et al., 2009). In that context, most of the proposed methods and metrics make use of (explicit or inferred) query aspects (topics or interpretations) to rank higher the most likely results (Demidova et al., 2010), or diversify a prior result set (Clarke et al., 2008; Agrawal et al., 2009; Chandar and Carterette, 2010; Radlinski et al., 2008; Rafiei et al., 2010).

In recommender systems diversity has been typically defined in an ad-hoc way, often mixing concepts such as diversity, novelty and coverage. For example, in (Salter and Antonopoulos, 2006) the authors make use of the catalog coverage defined above as a measure of recommendation diversity. A similar assumption is done in (Kwon, 2008). In (Zhou et al., 2010) the authors show that by tuning appropriately a hybrid recommender it is possible to obtain simultaneous gains in both accuracy and diversity, which is measured as the inter-list distance between every pair of users in the collection. Zhang and Hurley (2008) measure the novelty of an item by the amount of diversity it brings to the recommendation list, which is computed using a distance or dissimilarity function.

More formal definitions for diversity have also been proposed. In (Lathia et al., 2010) the authors propose to analyse diversity of top-N lists over time by comparing the intersection of sequential top-N lists. A statistical measure of diversity is proposed in (Zhang and Hurley, 2009), where the authors consider a recommendation algorithm to be fully diverse if it is equally likely to recommend any item that the user likes. In (Jambor and Wang, 2010b), the introduction of the covariance matrix into the optimisation problem leads to promote items in the long tail. A similar result is obtained in (Celma and Herrera, 2008), where the items with fewer interactions within the community of users (long tail) are assumed to be more likely to be unknown. Based on item similarities and focused on content-based algorithms, the authors in (Bradley and Smyth, 2001) propose a quality metric which considers both the diversity and similarity obtained in the recommendation list. A definition based on the entropy of the probability distributions of each recommender with respect to the items is proposed in (Bellogín et al., 2010), and the Gini's index is used in (Fleder and Hosanagar, 2009).

Finally, in (Vargas and Castells, 2011) a formal framework for the definition of novelty and diversity metrics is presented, where several previous metrics are unified

by identifying three ground concepts at the roots of novelty and diversity: choice, discovery, and relevance.

Other metrics such as serendipity, privacy, adaptivity, confidence, and scalability have been less explored in the literature, but their importance and application to recommender systems have already been discussed, making clear their relation with the user’s experience and satisfaction, which is the ultimate goal of a “good” recommender system (Herlocker et al., 2004; McNee et al., 2006; Shani and Gunawardana, 2011).

### 3.3 Experimental setup

An important decision in the experimental configuration of a recommender evaluation is the dataset partition strategy. How the datasets are partitioned into training and test sets may have a considerable impact on the final performance results, and may cause some recommenders to obtain better or worse results depending on how this partition is configured. Although an exhaustive analysis of the different possibilities to choose the ratings/items to be hidden is out of the scope of this thesis, we briefly discuss now some of the most well-known methods used.

First, we have to choose whether or not to take time into account (Gunawardana and Shani, 2009). Time-based approaches naturally require the availability of user interaction data timestamps. A simple approach is to select a time point in the available interaction data timeline to separate training data (all interaction records prior to that point) and test data (dated after the split time point). The split point can be set so as to, for instance, have a desired training/test ratio in the experiment. The ratio can be global, with a single common split point for all users, or user-specific, to ensure the same ratio per user. Time-based approaches have the advantage of more realistically matching working application scenarios, where “future” user likes (which would translate to positive response to recommendations by the system) are to be predicted based on past evidence. As an example, the well-known Netflix prize provided a dataset where the test set for each user consisted on her most recent ratings (Bennett and Lanning, 2007).

If we ignore time, there are at least the following three strategies to select the items to hide from each user: a) sample a fixed number (different) for each user; b) sample a fixed (but the same for all) number for each user, also known as *given n* or *all but n* protocols; c) sample a percentage of all the interactions using *cross-validation*. The most usual protocol is the last one (Goldberg et al., 2001; Sarwar et al., 2001), although several authors have also used the *all but n* protocol (Breese et al., 1998; Wang et al., 2008a). The MovieLens datasets provide random splits following a five-fold cross validation strategy, as we shall see in the next section.

Nonetheless, independently from the dataset partition, it is recognised that the goals for which an evaluation is performed may be different in each situation, and thus, a different setting (and partition protocol) should be developed (Herlocker et al., 2004; Gunawardana and Shani, 2009). If that is not the case, the results obtained in a particular setting would be biased and difficult to use in further experiments, for instance, in an online experimentation.

Furthermore, as mentioned earlier, in order to evaluate ranked recommendations for a target user  $u$ , it is required to select two sets of items, namely relevant and not relevant. In the next chapter, we describe different possibilities explored in the literature, along with a detailed analysis of these alternatives and the possible biases that may appear.

## 3.4 Evaluation datasets

In this section we present three datasets that were used in the experimental parts of this thesis. The datasets correspond to different domains: movie recommendation, in which user preferences are provided in the form of ratings, and music recommendation, where user preferences are derived from implicit (log-based) evidence. Furthermore, one of the datasets includes social information that can be exploited by social filtering algorithms.

### 3.4.1 MovieLens dataset

The GroupLens research lab<sup>1</sup> has released different datasets obtained from user interaction in the MovieLens recommender system. At the time of writing, there are three publicly available MovieLens datasets of different sizes:

- The 100K dataset, containing 100,000 ratings for 1,682 movies by 963 users.
- The 1M dataset, with one million ratings has 6,040 users and 3,900 movies.
- The 10M dataset, with 10 million ratings consists of almost 71,600 users and 10,700 movies, and 100,000 tag assignments.

Although there are larger public datasets (such as the one provided for the well-known competition organised by Netflix<sup>2</sup> between 2006 and 2009), the first two MovieLens datasets are currently, by far, the most used in the field.

The ratings range on a 5-star scale in all three datasets; the 100K and 1M versions only use “integer” stars, and 10M uses “half star” precision (ten discrete rating values). Every user has at least 20 ratings in any of the datasets.

---

<sup>1</sup> GroupLens research lab, <http://www.grouplens.org>

<sup>2</sup> Netflix site, <http://www.netflix.com>, and Netflix Prize webpage, <http://www.netflixprize.com>

### 3.4.2 Last.fm dataset

Last.fm is a social music website. At the moment, the site has more than 40 million users (claimed 30 million active in 2009<sup>3</sup>) in more than 190 countries. Several authors have analysed and used this system for research purposes; special mention deserves those who have made their datasets public, such as (Konstas et al., 2009), (Celma, 2010), and (Cantador et al., 2011).

In 2010, Óscar Celma released two datasets collected using the Last.fm API. The first one (usually referred to as 360K) contains the number of plays (called *scrobblings* in the platform) of almost 360,000 users, counted on music artists, amounting to more than 17 million of (user, artist, playcounts) tuples. The second dataset (named 1K) contains fewer users (nearly 1,000) but, in contrast to the previous one, the whole listening history of each user is collected as tuples (user, timestamp, artist, music track) for up to 19 million tuples.

### 3.4.3 CAMRa dataset

In 2010 the 1<sup>st</sup> Challenge on Context-aware Movie Recommendation (CAMRa 2010<sup>4</sup>) was held at the 4<sup>th</sup> ACM conference on Recommender Systems (RecSys 2010). The challenge organisers released four datasets that were used in three different challenge tracks (Adomavicius et al., 2010). These tracks were focused on temporal recommendation (*weekly recommendation*), recommendation based on mood (*Moviepilot track*), and social recommendation (*Filmtipset track*). Two different datasets were provided for the first track, whereas the second and third tracks were assigned a different dataset each (Said et al., 2010).

These datasets were gathered from the Filmtipset<sup>5</sup> and Moviepilot<sup>6</sup> communities, and, depending on the track, contained social links between users, movie ratings, movie reviews, review ratings, comments about actors and movies, movie directors and writers, lists of favourite movies, moods, and links between similar movies. Filmtipset is the largest online social community in the movie domain in Sweden, with more than 90,000 registered users and 20 million ratings in its database. Moviepilot, on the other hand, is the leading online movie and TV recommendation community in Germany; it has over 100,000 registered users and a database of over 40,000 movies with roughly 7.5 million ratings (Said et al., 2010).

Further editions of this challenge have also released datasets related to recommendation tasks (focused on group recommendation in 2011 (Said et al., 2011) and

<sup>3</sup> Announcement, <http://blog.last.fm/2009/03/24/lastfm-radio-announcement>

<sup>4</sup> CAMRa site, <http://2010.camrachallenge.com/>

<sup>5</sup> Filmtipset site, <http://www.filmtipset.se>

<sup>6</sup> Moviepilot site, <http://www.moviepilot.de>

on additional context information in 2012). However, they were not used in this thesis, and thus, they are not described in detail here.

### 3.5 Summary

In the Recommender Systems literature several evaluation metrics, protocols, and methodologies have been defined. It remains unclear the equivalence between them and the extent to which they would provide comparable results.

The problem of evaluating recommender systems has been a major object of study and methodological research in the field since its earliest days. Error-based metrics have widely dominated the field, and precision-based metrics have started to be adopted more recently. Other metrics from the Machine Learning field have been proposed but they are not widely used in the community yet. Moreover, metrics for additional dimensions such as novelty or diversity have also started to be researched in the last few years.

There are, still, important characteristics of the evaluation methodologies and metrics that remain unexplored. In contrast to the Information Retrieval community, where statistical analysis and eventual biases in the evaluation as a whole have been studied (Buckley et al., 2006; Aslam et al., 2006; Soboroff, 2004), there is a lack of such an analysis for recommender systems. This raises a key issue for our research which shall be analysed in depth in the next chapter, where we propose some alternative methodologies to overcome some of the possible biases that may arise.

# Chapter 4

## Ranking-based evaluation of recommender systems: experimental designs and biases

There is an increasing consensus in the Recommender Systems community that the dominant error-based evaluation metrics are insufficient, and to some extent inadequate, to properly assess the practical effectiveness of recommendations. Seeking to evaluate recommendation rankings – which largely determine the effective accuracy in matching user needs – rather than predicted rating values, Information Retrieval metrics have started to be applied to evaluate recommender systems.

In this chapter we analyse the main issues and potential divergences in the application of Information Retrieval methodologies on recommender system evaluation, and provide a systematic characterisation of experimental design alternatives for this adaptation. We lay out an experimental configuration framework upon which we identify and analyse specific statistical biases arising in the adaptation of Information Retrieval metrics to recommendation tasks, which considerably distort the empirical measurements, hindering the interpretation and comparison of results across experiments. We propose two experimental design approaches that effectively neutralise such biases to a large extent. We support our findings and proposals through both analytical and empirical evidence.

We start the chapter by introducing the problem of (un)biased evaluation in recommender systems. The remainder of the chapter follows by revisiting the principles and assumptions underlying the Information Retrieval evaluation methodology: the Cranfield paradigm (Section 4.2). After that, in Section 4.3 we elaborate a formal synthesis of the main approaches to the application of Information Retrieval metrics to recommendation. In Sections 4.4 and 4.5 we analyse, respectively, the sparsity and popularity biases of Information Retrieval metrics on recommendation tasks. We present and evaluate two approaches to avoid these biases in Section 4.6, and end with some conclusions in Section 4.7.

## 4.1 Introduction

There seems to be a raising awareness in the Recommender Systems (RS) community that important – or even central – open questions remain to be addressed concerning the evaluation of recommender systems. As we mentioned in the previous chapter, the error in predicting held-out user ratings has been by far the dominant offline evaluation methodology in the RS literature (Breese et al., 1998; Herlocker et al., 2004). The limitations of this approach are increasingly evident, and have been extensively pointed out (Cremonesi et al., 2010). The prediction error has been found to be far from enough or even adequate to assess the practical effectiveness of a recommender system in matching user needs. The end users of recommendations receive lists of items rather than rating values, whereby recommendation accuracy metrics – as surrogates of the evaluated task – should target the quality of the item selection and ranking, rather than the numeric system scores that determine this selection.

For this reason, researchers are turning towards metrics and methodologies from the Information Retrieval (IR) field (Barbieri et al., 2011; Cremonesi et al., 2010; Herlocker et al., 2004), where ranking evaluation has been studied and standardised for decades. Yet, gaps remain between the methodological formalisation of tasks in both fields, which result in divergences in the adoption of IR methodologies, hindering the interpretation and comparability of empirical observations by different authors. The use of IR evaluation techniques involves the adoption of the Cranfield paradigm (Voorhees and Harman, 2005), and common metrics such as precision, mean average precision (MAP), and normalised Discounted Cumulative Gain (nDCG) (Baeza-Yates and Ribeiro-Neto, 2011). Given the natural fit of top-n recommendation in an IR task scheme, the adoption of IR methodologies would seem straightforward. However, recommendation tasks, settings, and available datasets for offline evaluation involve subtle differences with respect to the common IR settings and experimental assumptions, which result in substantial biases to the effectiveness measurements that may distort the empiric observations and hinder comparison across systems and experiments.

Furthermore, how to measure the performance of a recommender system is a key issue in our research. The variability in the experimental configurations, and the observed statistical biases of the evaluation methodologies should be well understood, since we aim to predict the performance of a system. We should avoid the situation where a metric shows some source of noise together with the recommender's quality, since then a predictor capturing only that noise would appear as an (equivocal) effective performance predictor.

Taking up from prior studies on the matter (Cremonesi et al., 2010; Herlocker et al., 2004; Shani and Gunawardana, 2011; Steck, 2011), we revisit the methodological assumptions underlying IR metrics, and analyse the differences between Recom-

mender Systems and Information Retrieval evaluation settings and their implications. Upon this, we identify two sources of bias in IR metrics on recommender systems: data sparsity and item popularity. We characterise and study the effect of these two factors both analytically and empirically. We show that the value range of common IR metrics is determined by the density of the available user preference information, to such an extent that the measured values per se are not meaningful, except for the purpose of comparison within a specific experiment. Furthermore, we show that the distribution of ratings among items has a drastic effect on how different algorithms compare to each other. Finally, we propose and analyse two approaches to mitigate popularity biases on the measured ranking quality, providing theoretical and empirical evidence of their effectiveness.

## 4.2 Cranfield paradigm for recommendation

Information Retrieval evaluation methodologies have been designed, studied, and refined over the years under the so-called *Cranfield paradigm* (van Rijsbergen, 1989; Voorhees, 2002b). In the Cranfield paradigm, as e.g. typically applied in the TREC campaigns (Voorhees and Harman, 2005), information retrieval systems are evaluated on a dataset comprising a set of documents, a set of queries – referred to as *topics* and consisting of a description or representation of user information needs –, and a set of relevance judgments by human assessors – referred to as *ground truth*. The assessors manually inspect queries and documents, and decide whether each document is relevant or not for a query. Theoretically, each query-document pair should be assessed for relevance, which, for thousands or millions of documents, is obviously unfeasible. Therefore, a so-called *pooling* approximation is applied, in which the assessors actually inspect and judge just a subset of the document collection, consisting of the union of the top-n documents returned by a set of systems for each query. These systems for pooling are commonly the ones to be evaluated and compared, and n is called the *pooling depth*, typically ranging around 100 documents. While this procedure obviously misses some relevant documents, it has been observed that the degree of incompleteness is reasonably small, and the missing relevance does not alter the empiric observations significantly, at least up to some ratio between the pooling depth and the collection size (Buckley et al., 2007).

Whereas in a search system users may enter multiple queries, the recommendation task – in its classic formulation – typically considers a single “user need” per user, that is, a user has a set of cohesive preferences which defines her main interests. In this view a natural fit of recommendation in the Cranfield paradigm would take users – as an abstract construct – as the equivalent of queries in ad-hoc retrieval (the user need to be satisfied), and items as equivalent to documents (the objects to be retrieved and ranked), summarised in Table 4.1. A first obvious difference is that

| Task element                            | TREC ad-hoc retrieval task      | Recommendation task                      |
|---|---------------------------------|--|
| Information need expression             | Topic (query and description)   | User profile                             |
| Candidate answers                       | All documents in the collection | Target item set                          |
|   | Same for all queries            | One or more per user, commonly different |
| Document data available as system input | Document content                | Training ratings, item features          |
| Relevance                               | Topical, objective              | Personalised, subjective                 |
| Ground truth                            | Relevance judgments             | Test ratings                             |
| Relevance assessment                    | Editorial assessors             | End users                                |
| Relevance knowledge coverage            | Reasonably complete (pooling)   | Highly incomplete (inherently to task)   |

**Table 4.1. Fitting the recommendation task in the Cranfield IR evaluation paradigm**

queries are explicit representations of specific information needs, whereas in the recommendation setting, user profile records are a global and implicit representation of what the user may need or like. Still, the query-user mapping is valid, inasmuch as user profiles may rightfully fit in the IR scheme as “vague queries.”

The definition of ground truth is less straightforward. User ratings for items, as available in common recommendation datasets, are indeed relevance judgments of items for user needs. However, many recommendation algorithms (chiefly, collaborative filtering methods) require these “relevance judgments” as input to compute recommendations. The rating data withholding evaluation approach, pervasive in RS research, naturally fits here: some test ratings can be held out as ground truth and the rest be left as training input for the systems. Differently from TREC, here the “queries” and the relevance assessments are both entered by the same people: the end-users. Furthermore, how much data are taken for training and for ground truth is left open to the experiment designers, thus adding a free variable to be watched over as it significantly impacts the measurements.

On the other hand, whereas in the IR setting all the documents in the collection are candidate answers for all queries, the set of target items on which recommender systems are tested for each user need not be necessarily the same. As already described in the previous chapter, in general, the items with a test rating are included in the candidate set for the raters, though not necessarily in a single run (Cremonesi et al., 2010). Moreover, it is common to select further non-rated target items, but not necessarily all the items (Bellogín et al., 2011a). Furthermore, the items rated by a user in the training set are generally excluded from the recommendation to this user. The way these options are configured has a drastic effect on the resulting measurements, with variations in orders of magnitude (Bellogín et al., 2011a).

In addition to this, the coverage of user ratings is inherently much smaller in recommender systems’ datasets compared to TREC collections. The amount of un-

known relevance – which in TREC is assumed to be negligible – is pervasive in recommendation settings (it is in fact intrinsic for the task to make sense), to a point where some assumptions of the IR methodology may not hold, and the gap between measured and real metric values becomes so significant that a metric’s absolute magnitude may just lose any meaning. Still, such measurements may support comparative assessments between systems, as far as the bias is system-independent.

Finally, the distribution of relevance in the retrieval space displays popularity patterns that are absent in IR datasets. The number of users who like each item is very variable (typically long-tailed) in recommendation datasets, whereas in TREC collections very few documents are relevant for more than one query. We shall show that this phenomenon has a very strong effect not only on metric values, but more importantly on how systems compare to each other.

In order to provide a formal basis for our study we start by elaborating a systematic characterisation of design alternatives for the adaptation of IR metrics to recommender systems, taking into account prior approaches described in the literature, such as those presented in the previous chapter. This formal framework will help us to analyse and describe the measurement biases in the application of IR metrics to recommender systems, and study new approaches to mitigate them.

## 4.3 Experimental design alternatives

The application of Information Retrieval metrics to recommender systems evaluation has been studied by several authors in the field (Barbieri et al., 2011; Breese et al., 1998; Cremonesi et al., 2010; Herlocker et al., 2004; Shani and Gunawardana, 2011). We elaborate here an experimental design framework that aims to synthesise commonalities and differences between studies, encompassing prior approaches and supporting new variants upon a common methodological grounding. We formalise the different methodologies presented in the previous chapter, and provide an equivalence between both formulations.

In the following, given a rating set split into training and test rating sets, we say an item  $i \in \mathcal{I}$  is relevant for a user  $u \in \mathcal{U}$  if  $u$  rated  $i$  positively, and its corresponding rating falls in the test set. By positive rating we mean a value above some design-dependent threshold. All other items (non-positively rated or non-rated) are considered as non-relevant. Like in the previous chapter, recommender systems are requested to rank a set of target items  $T_u$  for each user. Such sets do not need to be the same for each user, and can be formed in different ways. In all configurations  $T_u$  contains a combination of relevant and non-relevant items, and the different approaches are characterised by how these are selected, as we describe next.

| Design settings      |          | Alternatives   |
|----------------------|----------|--|
| Base candidate items |          | <b>AI</b> $\mathcal{C} = \mathcal{I}$<br><b>TI</b> $\mathcal{C} = \bigcup_{u \in \mathcal{U}} R_{\text{test}}(u)$      |
| Item selection       | Relevant | <b>AR</b> $T_u \supset PR_{\text{test}}(u)$<br><b>IR</b> $ T_u^r \cap PR_{\text{test}}(u)  = 1$                        |
|                      |          | <b>AN</b> $N_u = \mathcal{C} - PR_{\text{test}}(u) - R_{\text{train}}(u)$<br><b>NN</b> Fixed $ N_u $ , random sampling |

Table 4.2. Design alternatives in target item set formation.

### 4.3.1 Target Item Sampling

We identify three significant design axes in the formation of the target item sets: candidate item selection, relevant item selection, and irrelevant item sampling. We consider two relevant alternatives for each of these axes, summarised in Table 4.2, which we describe next.

We shall use  $R(u)$  and  $PR(u)$  to denote the set of all and positively rated items by user  $u$ , respectively, and  $r(u)$ ,  $pr(u)$  to denote the respective size of those sets. With the subscripts “test” and “train” we shall denote the part of such sets (or their sizes) contained on the corresponding side of a data split. An equivalent notation  $r(i)$ ,  $pr(i)$ , and so on, will be used for the ratings of an item, and when no user or item is indicated, the total number of ratings is denoted. This notation and the rest to be used along the chapter are summarised in Table 4.3.

Let  $N_u = T_u - PR_{\text{test}}(u)$  be the non-relevant target items for  $u$ . As a general rule, we assume non-relevant items are randomly sampled from a subset of candidate items  $\mathcal{C} \subset \mathcal{I}$ , the choice of which is a design option. We mainly find two significant alternatives for this choice:  $\mathcal{C} = \mathcal{I}$  (e.g. (Shani and Gunawardana, 2011)) and  $\mathcal{C} = \bigcup_{u \in \mathcal{U}} R_{\text{test}}(u)$  (e.g. (Bellogín et al., 2011a; Vargas and Castells, 2011)). The first one, which we denote as **AI** for “all items”, matches the typical IR evaluation setting, where the evaluated systems take the whole collection as the candidate answers. The second, to which we shall refer as **TI** (“test items”) is an advisable condition to avoid certain biases in the evaluation of RS, as we shall see.

Once  $\mathcal{C}$  is set, for each user we select a set  $N_u \subset \mathcal{C} - PR_{\text{test}}(u) - R_{\text{train}}(u)$ .  $N_u$  can be sampled randomly for a fixed size  $|N_u|$  (we call this option **NN** for “N non-relevant”), or all candidate items can be included in the target set,  $N_u = \mathcal{C} - PR_{\text{test}}(u) - R_{\text{train}}(u)$  (we refer to this as **AN** for “all non-relevant”). Some authors have even used  $T_u = R_{\text{test}}(u)$  (Basu et al., 1998; Jambor and Wang, 2010a; Jambor and Wang, 2010b), but we discard this option as it results in a highly overestimated precision (Bellogín et al., 2011a). The size of  $N_u$  is thus a configuration parameter of

the experimental design. For instance, in (Cremonesi et al., 2010) the authors propose  $|N_u| = 1,000$ , whereas in (Bellogín et al., 2011a) the authors consider  $N_u = \sum_{v \in U} R_{\text{test}}(v) - R_{\text{train}}(u) - PR_{\text{test}}(u)$ , among other alternatives. To the best of our knowledge, the criteria for setting this parameter have not been analysed in detail in the literature, leaving it to common sense and/or trial and error. It is worth noting nonetheless that in general  $|N_u|$  determines the number of calls to the recommendation algorithms, whereby this parameter provides a handle for adjustment of the cost of the experiments.

Regarding the relevant item selection, two main options are reported in the literature, to which we shall refer as **AR** for “all relevant”, and **1R** for “one relevant.” In the AR approach all relevant items are included in the target set, i.e.,  $T_u \supseteq PR_{\text{test}}(u)$  (Bellogín et al., 2011a). In the 1R approach, for user  $u$ , several target item sets  $T_u^r$  are formed, each including a single relevant item (Cremonesi et al., 2010). This approach may be more sensitive to the lack of recommendation coverage, as we shall observe later on. The choice between an AR or a 1R design involves a difference in the way the ranking quality metrics are computed, as we shall discuss in the next section.

| Symbol            | Meaning   |   |
|-------------------|---|---|
| $U$               | $\mathcal{I}$   | $\mathcal{C}$   |
| $r$               | $r_{\text{test}}$                                     | $r_{\text{train}}$  |
| $pr$              | $pr_{\text{test}}$                                    | $pr_{\text{train}}$   |
| $R(u)$            | $R_{\text{test}}(u)$                                  | $R_{\text{train}}(u)$   |
| $PR(u)$           | $PR_{\text{test}}(u)$                                 | $PR_{\text{train}}(u)$  |
| $r(u)$            | $pr(u)$   | ...   |
| $r(i)$            | $pr(i)$   | ...   |
| $T_u$             | $T_u^r$   | Set of target items for $u$ in AR   1R                                      |
| $N_u$             | $N_u^r$   | Non-relevant items added to build $T_u$   $T_u^r$                           |
| $P_s^u @ n(T_u)$  | $P@n$ of item set $T_u$ as ranked by $s$ for $u$      |   |
| $top_s^u(T_u, n)$ | Top $n$ items in $T_u$ as ranked by $s$ for $u$       |   |
| $\tau_s^u(i, S)$  | Position of $i$ in $S \ni i$ as ranked by $s$ for $u$ |   |
| $i_k^{u,s}$       | $i_k^{u,r,s}$   | The item ranked $k$ -th in $T_u$   $T_u^r$ by $s$ for $u$                   |
| $\sigma$          | Split ratio: $r_{\text{test}}/r$                      |   |
| $\rho_u$          | $\rho$  | $\rho_u =  T_u \cap PR_{\text{test}}(u) / T_u , \rho = \text{avg}_u \rho_u$ |
| $t$               | “Average” target set size: $1/\text{avg}_u(1/ T_u )$  |   |
| $\delta$          | Relevance density in target sets: $pr/(t U )$         |   |

Table 4.3. Notation summary.

### 4.3.2 AR vs. 1R Precision

Essentially, the way metrics are defined in AR and 1R differs in how they are averaged. In AR the metrics are computed on each target set  $T_u$  in the standard way as in IR, and then averaged over users (as if they were queries). As a representative and simple to analyse metric, we shall use  $P@n$  henceforth, but similar properties to all the ones discussed here are observed for other metrics such as MAP and nDCG. The mean AR precision of a recommender system  $s$  can be expressed as:

$$P_s@n = \frac{1}{|U|} \sum_{u \in U} \frac{1}{n} |top_s^u(T_u, n) \cap PR_{test}(u)|$$

where  $top_s^u(T_u, n)$  denotes the top  $n$  items in  $T_u$  ranked by  $s$  for  $u$ .

In the 1R design, drawing from (Cremonesi et al., 2010), we compute and average the metrics over the  $T_u^r$  sets, as follows:

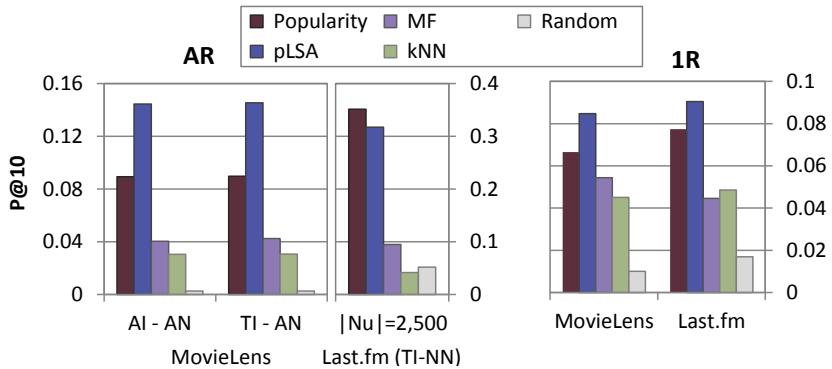
$$1RP_s@n = precision(n) = \frac{1}{pr_{test}} \sum_{u \in U} \sum_{r=1}^{pr_{test}(u)} P_s^u@n(T_u^r) \quad (4.1)$$

where  $P_s^u@n(T_u^r)$  is the standard precision of  $T_u^r$  for  $u$ . This form to express the metric is equivalent to the original formulation in (Cremonesi et al., 2010), but lets a straightforward generalisation to any other IR metric such as MAP and nDCG, by just using them in place of  $P_s^u@n$  in Equation (4.1). We shall intentionally use the same symbol  $P$  to refer both to 1R and AR precisions when there is no ambiguity. Whenever there may be confusion, or we wish to stress the distinction, we shall use 1RP to explicitly denote 1R precision.

AR precision basically corresponds to the standard precision as defined in IR, whereas 1R precision, while following essentially the same principle, departs from it in the formation of runs, and the way to average values. Additionally, note that the maximum value of 1RP@n is  $1/n$  as we shall see in the next section, mainly since each run has only one relevant item. Besides, in Section 4.4 we shall establish a formal relation between both ways to compute precision.

### 4.3.3 Preliminary Test

In order to illustrate the effects of the different described alternatives, we show their results on three common collaborative filtering algorithms, based respectively on probabilistic Latent Semantic Analisys (pLSA) (Hofmann, 2004), matrix factorisation (MF) (Koren et al., 2009), and user-based nearest-neighbours (kNN) (Cremonesi et al., 2010). As additional baselines, we include recommendation by popularity and random recommendation. We use two datasets: the 1M version of MovieLens, and



**Figure 4.1. Precision of different recommendation algorithms on MovieLens 1M and Last.fm using AR and 1R configurations.**

an extract from Last.fm published by Ó. Celma (Celma and Herrera, 2008). Details about the implementation and datasets partition are provided in Appendix A.

Figure 4.1 shows the P@10 results with AR and 1R configurations. For 1R we shall always use TI-NN, with  $|T_u| = 100$ . This is a significantly lower value than  $|T_u| = 1,001$  reported in (Cremonesi et al., 2010), but we have found it sufficient to ensure statistical significance (e.g. Wilcoxon  $p \ll 0.001$  for all pairwise differences between the recommenders in Figure 4.1), at a considerably reduced execution cost. We adopt the TI policy in 1R to avoid biases that we shall describe later. In the AR configuration we show TI-AN and AI-AN for MovieLens, though we shall generally stick to TI-AN in the rest of the chapter. In Last.fm we use only TI-NN and a temporal split, with  $|N_u| = 2,500$  for efficiency reasons, since  $|\mathcal{I}| = 176,948$  is considerably large in this dataset. We set the positive relevance rating threshold to 5 in MovieLens, as in (Cremonesi et al., 2010), whereas in Last.fm, we take any number above 2 playcounts as a sign of positive preference. We have experimented with other thresholds for positive ratings, obtaining equivalent results to all the ones that are reported here – the only difference is discussed in Section 4.6.

It can be seen that pLSA consistently performs best in most experimental configurations, closely followed by popularity, which is the best approach in Last.fm with AR, and that MF is generally superior to kNN. Some aspects strike our attention. First, even though P@10 is supposed to measure the same thing in all cases, the range of the metric varies considerably across configurations and datasets, and even the comparison is not always consistent. For instance, in AR popularity ranges from 0.08 on MovieLens to 0.35 on Last.fm; and AR vs. 1R produces some disagreeing comparisons on Last.fm. It may also be surprising that popularity, a non-personalised method, fares so well compared to other algorithms. This effect was already found recently in (Cremonesi et al., 2010) and (Steck, 2011). We also see that TI and AI produce almost the same results. This is because  $\bigcup_{u \in \mathcal{U}} R_{test}(u) \sim \mathcal{I}$  in MovieLens; differences become noticeable in configurations where  $\bigcup_{u \in \mathcal{U}} R_{test}(u)$  is significantly

smaller than  $\mathcal{I}$ , as we shall see in Section 4.6.2. As mentioned before, note that in this case, the upper bound of P@10 for the 1R methodology is 0.10.

Some of this variability may reflect actual strengths and weaknesses of the algorithms for different datasets, but we shall show that a significant part of the observed variations is due to statistical biases arising in the adaptation of the Cranfield methodology to recommendation data, and are therefore meaningless with respect to the assessment of the recommenders' accuracy. Specifically, we have found that the metrics are strongly biased to test data sparsity and item popularity. We shall analyse this in detail in Sections 4.4 and 4.5, but before that we establish a relation between AR and 1R precision that will help in this analysis.

#### 4.3.4 Relation between AR and 1R

We have seen that AR and 1R precisions produce in general quite different values, and we shall show they display different dependencies over certain factors. We find nonetheless a direct relation between the two metrics. Specifically, 1R precision is bound linearly by NN-AR precision, that is,  $1RP_s@n = \Theta(P_s@n)$ , as we show next.

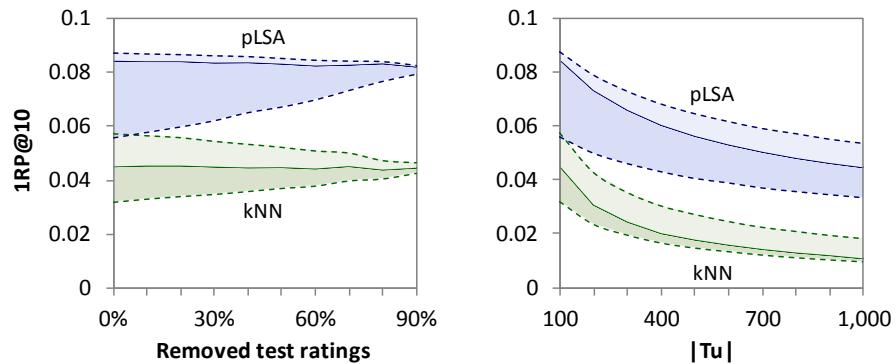
**Lemma.** Let us assume the irrelevant item sampling in 1R is done only once for all the test ratings of a user, that is, we select the same set of non-relevant items  $N_u^r = N_u$  in the  $T_u^r$  target sets. If we denote  $T_u = N_u \cup PR_{test}(u)$  – in other words,  $T_u = \bigcup_r T_u^r$  –, we have:

$$\frac{|\mathcal{U}|P_s@n}{pr_{test}} \leq 1RP_s@n \leq \frac{\sum_{u \in \mathcal{U}} m_u P_s^u @ m_u(T_u)}{n \cdot pr_{test}} \quad (4.2)$$

with  $m_u = n + pr_{test}(u) - 1$ , where  $P_s@n$  is the NN-AR precision computed with the target sets  $\{T_u\}$ .

**Proof.** Let  $i_u^r$  be the relevant item included in  $T_u^r$ , and let  $\tau_s^u(i, S)$  denote the ranking position assigned to  $i$  by  $s$  for  $u$  within a set  $S$ , where  $i \in S$ . Since  $T_u^r \subset T_u$ , we have that  $\tau_s^u(i_u^r, T_u^r) \leq \tau_s^u(i_u^r, T_u)$ . This means that if  $i_u^r$  is ranked above  $n$  in  $T_u$ , then it is also above  $n$  in its target set  $T_u^r$ . Hence  $\sum_{r=1}^{np_{test}(u)} |\text{top}_s^u(T_u^r, n) \cap PR_{test}(u)| \geq |\text{top}_s^u(T_u, n) \cap PR_{test}(u)|$ . Summing on  $u$ , and dividing by  $n$  and  $pr_{test}$  we prove the first inequality of Equation (4.2).

On the other hand, it is easy to see that  $\tau_s^u(i_u^r, T_u^k) \geq \tau_s^u(i_u^r, T_u) + pr_{test}(u) - 1$ . Thus, if  $i_u^r$  is ranked above  $n$  in  $T_u^k$ , then it is above  $m_u = n + pr_{test}(u) - 1$  in  $T_u$ . Thus  $\sum_{r=1}^{np_{test}(u)} |\text{top}_s^u(T_u^r, n) \cap PR_{test}(u)| \leq |\text{top}_s^u(T_u, m_u) \cap PR_{test}(u)| = m_u P_s@m_u(T_u)$ . And the second inequality of Equation (4.2) follows again by summing on  $u$ , and dividing by  $n$  and  $pr_{test}$ .  $\square$



**Figure 4.2. Empiric illustration of Equation (4.2).** The curves show 1RP@10 and its bounds, for pLSA and kNN over MovieLens 1M. The light and dark shades mark the distance to the upper and lower bounds, respectively. The left side shows the evolution when progressively removing test ratings, and the right side displays the variation with  $|T_u|$  ranging from 100 to 1,000.

Note that the assumption  $N_u^r = N_u$  in the lemma is mild, inasmuch as the statistical advantage in taking different  $N_u^r$  for each  $r$  is unclear. Even in that case,  $P_s @ n$  and  $\text{avg}_{u \in U} (m_u P_s^u @ m_u (T_u))$  should be reasonably stable with respect to the random sampling of  $N_u^r$ , and thus Equation (4.2) tends to hold. Figure 4.2 illustrates the relation between the AR bounds and the 1R values. The empiric observation suggests they provide similar while not fully redundant assessments. We also see that the bounding interval reduces progressively as  $|T_u|$  is increased (right), and even faster with test data sparsity (left) – in sum, the metric converges to its bounds as  $|T_u| \gg \text{avg}_u pr_{test}(u) = pr_{test}/|U|$ .

### 4.3.5 Limitations of error-based metrics

The analysis presented in (Bellogín et al., 2011a) leads to question again the suitability of error metrics. As in (McLaughlin and Herlocker, 2004), we found that there is no direct equivalence between results with error- and precision-based metrics. Common sense suggests that putting more relevant items in the top-N is more important for real recommendation effectiveness than being accurate with predicted rating values, which are usually not even shown to real users. Our study confirms that measured results differ between these two perspectives. An online experiment, where real users’ feedback is contrasted to the theoretic measurements, may shed further light for an objective assessment and finer analysis of which methodology better captures user satisfaction.

Furthermore, the use of error-based metrics may not be applicable depending on the dataset or the recommender evaluated. For instance, log-based datasets and probabilistic (e.g. pLSA) or popularity-based recommenders cannot be evaluated using error-based metrics because no real ratings are available in the first case, and in

the second case because such recommenders do not necessarily predict a rating, not even a score in the range of ratings (Cremonesi et al., 2010).

The application of ranking-based metrics to recommendation, nonetheless, is far from being trivial. Firstly, there are obvious differences between the Cranfield paradigm and a standard recommendation context, as described in Section 4.2. Secondly, the evaluation methodology may be sensitive to any statistical bias which may appear in the process. In the next sections we shall analyse two of these sources of bias: sparsity and popularity.

## 4.4 Sparsity bias

As mentioned earlier, we identify two strong biases in precision metrics when applied to recommendation. The first one is a sensitivity to the ratio of the test ratings vs. the added non-relevant items. We study this effect by an analysis of the expected precision for non-personalised and random recommendations in the AR and 1R settings.

### 4.4.1 Expected Precision

Let  $i_k^{u,s} \in T_u$  be the item ranked at position  $k$  in the recommendation output for  $u$  by a recommender system  $s$ , and let  $\sigma$  be the ratio of test data in the training-test data split. In an AR setup the expected precision at  $n$  (over the sampling space of data splits with ratio  $\sigma$ , the sampling of  $N_u$ , and any potential non-deterministic aspect of the recommender system – as e.g. in a random recommender) is:

$$E[P_s@n] = \text{avg}_{u \in U} \left( \frac{1}{n} \sum_{k=1}^n p(\text{rel}|i_k^{u,s}, u, T_u) \right)$$

where  $p(\text{rel}|i, u)$  denotes the probability that item  $i$  is relevant for user  $u$ , i.e., the probability that  $i \in PR_{test}(u)$ . Now we may write  $p(\text{rel}|i_k^{u,s}, u, T_u) \stackrel{\text{def}}{=} p(\text{rel}, T_u | i_k^{u,s}, u) / p(T_u | i_k^{u,s}, u)$ , where we have  $p(T_u | i_k^{u,s}, u) = p(i_k^{u,s} \in T_u) = |T_u|/|\mathcal{C}|$ . On the other hand,  $p(\text{rel}, T_u | i_k^{u,s}, u) \stackrel{\text{def}}{=} p(i_k^{u,s} \in PR_{test}(u) \cap T_u) = p(i_k^{u,s} \in PR_{test}(u)) = p(\text{rel}|i_k^{u,s}, u)$ , since  $PR_{test}(u) \subset T_u$  in the AR methodology. If  $s$  is a non-personalised recommender then  $i_k^{u,s}$  and  $u$  are mutually independent, and it can be seen that  $\text{avg}_{u \in U} p(\text{rel}|i_k^{u,s}, u) = \text{avg}_{u \in U} p(\text{rel}|i_k^{u,s})$ . All this gives:

$$E[P_s@n] = \frac{|\mathcal{C}|}{n \cdot t} \sum_{k=1}^n \text{avg}_{u \in U} p(\text{rel}|i_k^{u,s})$$

where  $1/t = \text{avg}_u(1/|T_u|)$  – if  $T_u$  have all the same size, then  $t = |T_u|$ . As all relevant items for each user are included in her target set, we have  $p(\text{rel}|i_k^{u,s}) = E[\text{pr}_{\text{test}}(i_k^{u,s})]/|\mathcal{U}|$ . If ratings are split at random into test and training, this is equal to  $\sigma \cdot \text{pr}(i_k^{u,s})/|\mathcal{U}|$ . Hence, we have:

$$E[P_s@n] = \frac{\sigma |\mathcal{C}|}{n \cdot t |\mathcal{U}|} \sum_{k=1}^n \text{avg}_{u \in \mathcal{U}} \text{pr}(i_k^{u,s}) \quad (4.3)$$

Now, if items were recommended at random, we would have  $E[\text{pr}(i_k^{u,RND})] = \text{pr}/|\mathcal{I}|$ , and therefore:

$$E[P_{RND}@n] = E[P_{RND}] \sim \frac{\sigma \cdot \text{pr}}{t |\mathcal{U}|} = \sigma \cdot \delta \quad (4.4)$$

where  $\delta$  is the average density of known relevance – which depends on how many preferences for items the users have conveyed, and the size of the target test item sets.

On the other hand, in a 1R evaluation setup, we have:

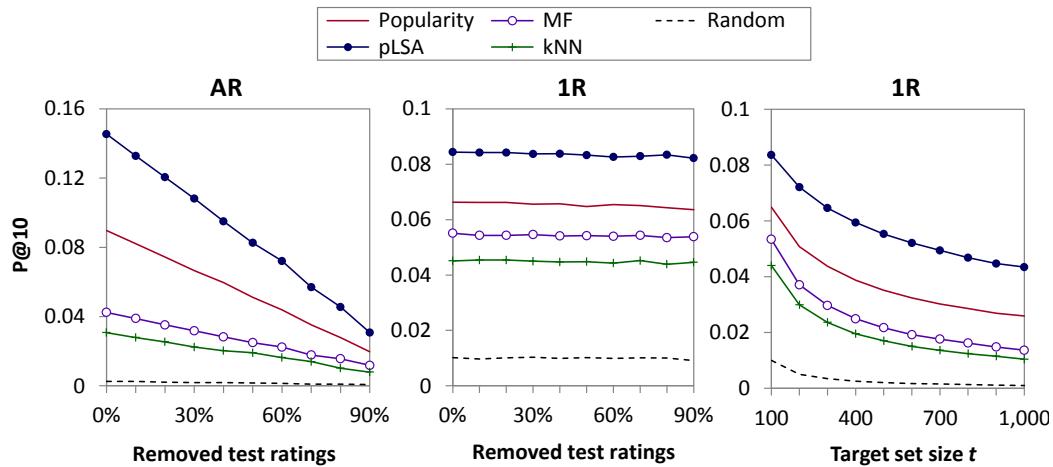
$$E[1RP_s@n] = \frac{1}{n \cdot \text{pr}_{\text{test}}} \sum_{u \in \mathcal{U}} \sum_{r=1}^{\text{pr}_{\text{test}}(u)} \sum_{k=1}^n p(\text{rel}|i_k^{u,r,s}, u, T_u^r)$$

where  $i_k^{u,r,s} \in T_u^r$  denotes the item ranked at position  $k$  in  $T_u^r$ . For random recommendation, we have  $p(\text{rel}|i_k^{u,r,RND}, u, T_u^r) = 1/|T_u^r| = 1/t$  since all target sets have the same size, whereby we have:

$$E[1RP_{RND}@n] = E[1RP_{RND}] = 1/t \quad (4.5)$$

#### 4.4.2 Test Sparsity Bias

The above results for the expected random precision provide a formal insight on strong metric biases to characteristics of the data and the experimental configuration. In both Equations (4.4) for AR and (4.5) for 1R, we may express the expected random precision as  $E[P_{RND}@n] = \text{avg}_u \rho_u = \rho$ , where  $\rho_u$  is the ratio of positively rated items by  $u$  in  $T_u$  (or  $T_u^r$ , for that matter), and  $\rho \sim \sigma \cdot \delta$ , or  $\rho = 1/t$ , depending on the experimental approach. In the AR approach the density  $\delta$ , and thus the  $\rho$  ratio, are also inversely proportional to  $t$ . Precision in this methodology is therefore sensitive to (grows linearly with)  $\sigma$  and  $\text{pr}$ , and is inversely proportional to  $t$ , whereas 1R is only sensitive (inversely proportional) to  $t$ . The expected precision of random recommendation naturally provides a lower bound for any acceptable recommender. Note that in any configuration of AR and 1R, the total precision of any system is  $P_s = P_{RND} = \rho = E[P_{RND}@n]$ , since as all systems are required to return



**Figure 4.3. Evolution of the precision of different recommendation algorithms on MovieLens 1M, for different degrees of test sparsity. The x axis of the left and center graphics shows different amounts of removed test ratings. The x axis in the right graphic is the size of the target item sets.**

(recommend) all items in the target sets  $T_u$  (or  $T_u^r$ ), that is, the total precision does not depend on the ranking. At lower cutoffs, we expect to have  $P_s@n > E[P_{RND}@n] = \rho$ . In other words, the lower bound – and so the expected range – for the  $P@n$  of recommender algorithms grows with the average ratio of relevant items per target item set.

The  $\rho$  ratio – hence the random precision – thus depends on several aspects of the experimental setup (the experimental approach, the split ratio  $\sigma$ , the number of non-relevant items in the target sets), and the test collection (the number of ratings, the number of users). Therefore, since  $\rho$  and the random precision can be adjusted arbitrarily by how the test sets are split, constructed, etc., we may conclude that **the specific value of the metric has a use for comparative purposes, but has no particular meaning by itself, unless accompanied by the corresponding average relevance ratio  $\rho$  of the target test sets**. This is naturally in high contrast to common IR datasets, where both the document collection and the relevance information are fixed and not split or broken down into subsets. In fact, the metric values reported in the TREC campaigns have stayed within a roughly stable range over the years (Armstrong et al., 2009a; Armstrong et al., 2009b). Note also that the sparsity bias we analyse here is different from the impact of training data sparsity in the performance of collaborative filtering systems. What we describe is a statistical bias caused by the sparsity of test data (as a function of overall data sparsity and/or test data sampling), and its effect does not reflect any actual variation whatsoever in the true recommendation accuracy.

The sparsity bias explains the precision range variations observed earlier in Figure 4.1. The empirically obtained values of random precision match quite exactly the

theoretically expected ones. To what extent the random recommendation analysis generalises to other algorithms can be further analysed empirically. Figure 4.3 illustrates the bias trends over rating density and target set size, using the experimental setup of Section 4.3.3 (with TI-AN in AR, and TI-NN in 1R). We show only the results in MovieLens – they display a similar effect on Last.fm. In the left and center graphics, we simulate test sparsity by removing test ratings. In the right graphic we vary  $t = |T_u|$  in a 1R configuration. We observe that the empirical trends confirm the theoretical analysis: precision decreases linearly with density in the AR methodology (left graphic, confirming a linear dependence on  $\delta$ ), whereas precision is independent from the amount of test ratings in the 1R approach (center), and shows inverse proportionality to  $t$  (right). It can furthermore be seen that the biased behavior analytically described for random recommendation is very similarly displayed by the other recommenders (only differing in linear constants). This would confirm the explanatory power of the statistical trend analysis of random recommendation, as a good reference for similar biases in other recommenders. On the other hand, even though the precision values change drastically in magnitude, it would seem that the comparison between recommenders is not distorted by test sparsity. We find other biases in precision measurements, however, which do affect the comparison of recommenders, as we study in the next section.

## 4.5 Popularity bias

Sparsity is not the only bias the metric measurements are affected by. The high observed values for a non-personalised method such as recommendation by popularity raise the question of whether this really reflects a virtue of the recommender, or some other bias in the metric. We seek to shed some light on the question by a closer study.

### 4.5.1 Popularity-Driven Recommendation

Even though they contradict the personalisation principle, the good results of popularity recommendation can be given an intuitive explanation. By averaging over all users, precision metrics measure the overall satisfaction of the user population. A method that gets to satisfy a majority of users is very likely to perform well under such metrics. In other words, average precision metrics tend to favour the satisfaction of majorities, regardless of the dissatisfaction of minorities, whereby algorithms that target majority tastes will expectably yield good results on such metrics. This implicitly relies on the fact that on a random item split, the number of test ratings for an item correlates with its number of training ratings, and its number of positive ratings correlates with the total number of ratings. More formally, the advantage of

popularity-oriented recommendation comes from the fact that in a random rating split,  $E[pr_{test}(i)] \propto pr(i) \propto E[pr_{train}(i)] \propto r_{train}(i)$ , which means that the items with many training ratings will tend to have many positive test ratings, that is, they will be liked by many users according to the test data. We analyse this next, more formally and in more detail.

In a popularity recommender  $i_k^{u,POP}$  is the  $k$ -th item in the target set with most ratings in the training set – i.e., the system ranks items by decreasing order of  $r_{train}(i_k^{u,POP})$ . This ranking is almost user-independent (except for those, statistically negligible, user items already in training which are excluded from the ranking) and therefore, for an AR experimental design, Equation (4.3) applies. Since we have  $\sum_{k=1}^n pr(i_k^{u,POP}) = \max_s \sum_{k=1}^n pr(i_k^{u,s})$  (as far as  $E[pr_{test}(i)] \propto r_{train}(i)$  for a random training-test split), the popularity recommendation is the best possible non-personalised system, maximising  $E[P_s@n]$ . Popularity thus achieves a considerably high precision value, just for statistical reasons.

For a 1R experimental design, using Equation (4.2) (lemma) we have:

$$\frac{|\mathcal{U}|E[P_s@n]}{pr_{test}} \leq E[1RP_s@n] \leq \frac{\sum_u E[m_u P_s^u @ m_u(T_u)]}{n \cdot pr_{test}}$$

Now, since  $P_s@n$  and  $P_s^u@m_u$  above are computed by AR, we may elaborate from Equation (4.3) for a non-personalised recommender, and we get:

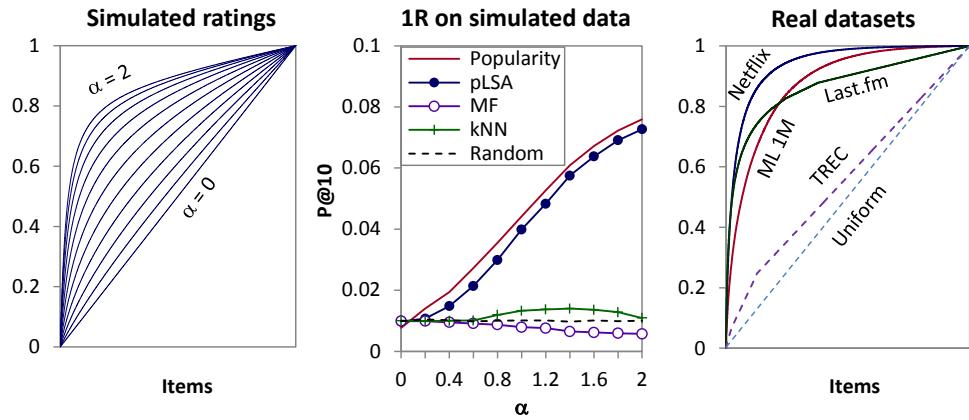
$$\frac{|\mathcal{I}|}{n \cdot t \cdot pr} \text{avg}_{u \in \mathcal{U}} \sum_{k=1}^n pr(i_k^{u,s}) \leq E[1RP_s@n] \leq \frac{|\mathcal{C}|}{n \cdot t \cdot pr} \text{avg}_{u \in \mathcal{U}} \sum_{k=1}^{m_u} pr(i_k^{u,s})$$

This experimental approach is thus equally biased to popular items, since the latter optimise  $\sum_{k=1}^n pr(i_k^{u,s})$ .

Note that the advantage of popularity over other recommenders is highly dependent on the skewness in the distribution of ratings over items: if all items were equally popular, the popularity recommender would degrade to random recommendation – in fact slightly worse, as  $pr_{test}(i) \propto r_{test}(i) = r/|\mathcal{I}| - r_{train}(i)$ , so popular items would have fewer positive test ratings. On the other extreme, if a few items (less than  $n$ ) are liked by most users, and the rest are liked by very few, then popularity approaches the maximum precision possible.

### 4.5.2 Popularity Distributions

In order to illustrate how the dependence between the popularity precision and the background popularity distribution evolves, we simulate different degrees of skewness in rating distributions. As a simulated distribution pattern we use a shifted power law  $r(i_k) = c_1 + \beta(c_2 + k)^{-\alpha}$ , where  $\alpha$  determines the skewness (e.g.  $\alpha \sim 1.4$  for MovieLens 1M). Figure 4.4 (left) shows the shape of generated distributions ranging



**Figure 4.4. Effect of popularity distribution skewness on the popularity bias.** The left graphic shows the cumulated popularity distribution of artificial datasets with simulated ratings, with skewness ranging from  $\alpha = 0$  to 2. The x axis represents items by popularity rank, and the y axis displays the cumulative ratio of ratings. The central graphic shows the precision of different recommendation algorithms on each of these simulated datasets. The right graphic shows the cumulative distribution of positive ratings in real datasets.

from uniform ( $\alpha = 0$ ) to a very steep long-tailed popularity distribution ( $\alpha = 2$ ), and (center) how the measured precision evolves in this range. The artificial data are created with the same number of users, items, and ratings (therefore the same rating density) as in MovieLens 1M, setting  $c_1$  and  $c_2$  by a fit to this dataset, and enforcing these constraints by adjusting  $\beta$ . The rating values are assigned randomly on a 1-5 scale, also based on the prior distribution of rating values in Movie-Lens.

The results in Figure 4.4 (center) evidence the fact that the precision of popularity-based recommendation is heavily determined by the skewness of the distribution. It benefits from steep distributions, and degrades to slightly below random (0.0077 vs. 0.0100) when popularity is uniform. This slightly below-random performance of popularity recommendation at  $\alpha = 0$  is explained by the fact that  $E[pr_{test}(i)] \propto E[r_{test}(i)] = r(i) - E[r_{train}(i)]$  is inverse to the popularity ranking by  $r_{train}(i)$  when  $r(i)$  is uniform, as predicted at the end of the previous section. kNN and MF stay essentially around random recommendation. This is because the data are devoid of any consistent preference pattern (as collaborative filtering techniques would assume) in this experiment, since the ratings are artificially assigned at random, and the results just show the “pure” statistical dependency to the popularity distribution. pLSA does seem to take advantage of item popularity, as it closely matches the effectiveness of popularity recommendation. We show only the 1R design, but the effect is the same in AR.

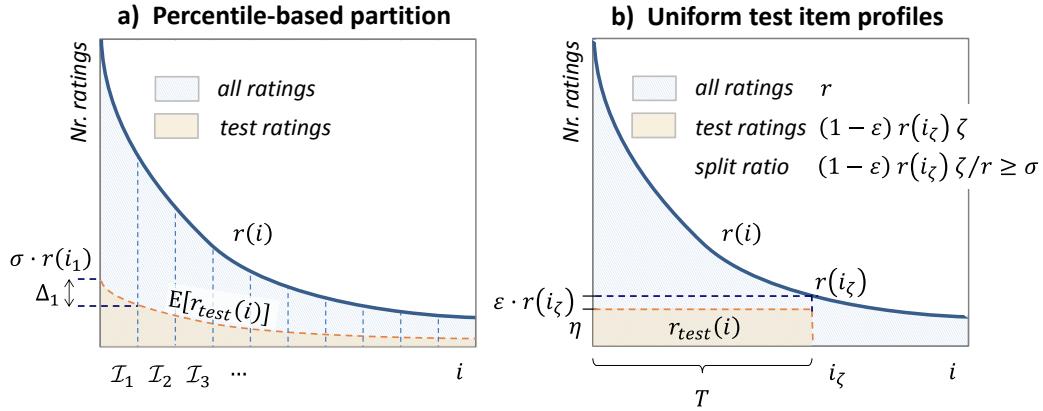
This observation also explains the difference between datasets from IR and those from recommendation with regards to the popularity bias. Figure 4.4 (right) shows the cumulative distribution of positive user interaction data per item in three

datasets: Netflix, MovieLens, and Last.fm (the dataset in Section 4.3.3). The shapes of the curves are typical of long-tailed distributions, where a few popular items accumulate most of the preference data (Celma, 2010; Celma and Cano, 2008). This contrasts with the distribution of positive relevance judgments over documents in TREC data (same figure) – where we have aggregated 30 individual tracks, filtering out the documents that are not relevant to any query, and obtaining a set of 703 queries, 129,277 documents, and 149,811 positive judgments. The TREC distribution is considerably flatter, not far from uniform: 87.2% of documents are relevant to just one query, and the maximum number of positive assessments per document is 25 (3.6% of queries), whereas the top popular item in Netflix, MovieLens, and Last.fm, is liked by 20.1%, 32.7% and 73% of users, respectively.

Several reasons account for this difference between retrieval and recommender datasets. First, in IR queries are selected by design, intending to provide a somewhat varied testbed to compare retrieval systems. Hence, including similar queries with overlapping relevance would not make much sense. Second, queries in natural search scenarios are generally more specific and narrower than global user tastes for recommendation, whereby the corresponding relevant sets have much less intersection. Furthermore, the TREC statistics we report are obtained by aggregating the data of many tracks, in order to seek any perceptible popularity slant. The typical TREC experiments are actually run on separate tracks comprising typically 50 queries, where very few documents, if any, are relevant to more than one query. Note also that even though we have filtered out over 0.7 million non-relevant plus nearly 5 million unlabeled documents in the TREC statistics, the non-relevant documents actually remain as input to the systems, contrarily to experiments in the recommender domain, thus making up an even flatter relevance distribution. Moreover, in the usual IR evaluation setting, the systems have no access to the relevance data – thus, they have no means to take a direct bias towards documents with many judgments –, whereas in recommendation, this is the primary input the systems (particularly collaborative filtering recommenders) build upon. The popularity phenomenon has therefore never been an issue in IR evaluation, and neither the metrics nor the methodologies have had to even consider this problem, which arises now when bringing them to the recommendation setting – where the overlap between user preferences is not only common, but actually needed by collaborative filtering algorithms.

## 4.6 Overcoming the popularity bias

After analysing the effects of popularity in precision metrics, the issue remains: to what extent do the good results of popularity recommendation reflect only a statistical bias in a metric, or any degree of actual recommendation quality? The same question should be raised for pLSA, which seems to follow the popularity trends quite



**Figure 4.5.** Rating splits by a) a popularity percentile partition (left), and b) a uniform number of test ratings per item (right). On the left, the red dashed split curve represents  $E[pr_{test}(i)]$  – i.e., the random split ratio needs not be applied on a per-item basis – whereas on the right it does represent  $pr_{test}(i)$ .

closely. We address the question by proposing and examining alternative experimental configurations, where the statistical role of popularity gets reduced, as we propose next.

### 4.6.1 Percentile-Based Approach (P1R)

We propose a first approach to neutralise the popularity bias, which consists in partitioning the set of items into  $m$  popularity percentiles  $\mathcal{I}_k \subset \mathcal{I}$ , breaking down the computation of accuracy by such percentiles, and averaging the  $m$  obtained values. By doing so, in a common long-tailed popularity distribution, the margin for the popularity bias is considerably reduced, as the difference  $\Delta_k$  in the number of positive test ratings per item between the most and least popular items of each percentile is not that high. The popularity recommender is forced to recommend as many unpopular as popular items, thus leveling the statistical advantage to a significant extent. It remains the optimal non-personalised algorithm, but the difference – and thus the bias – is considerably reduced. The technique is illustrated in Figure 4.5a.

A limitation of this approach is that it restricts the size of the target sets by  $|T_u| \leq |\mathcal{I}|/m$ . For instance, for  $m = 10$  in MovieLens 1M, this imposes a limit of  $|T_u| \leq \sim 370$ , which seems acceptable for 1R. The restriction can be more limiting in the AR approach, e.g. the TI and AI options cannot be applied (except within the percentiles). For this reason, we will only apply the percentile technique in the 1R design, a configuration to which we shall refer as **P1R**.

### 4.6.2 Uniform Test Item Profiles (UAR, U1R)

We now propose a second technique consisting of the formation of data splits where all items have the same amount of test ratings. The assumption is that the items with a high number of training ratings will no longer have a statistical advantage by having more positive test ratings. That is, the relation  $E[pr_{test}(i)] \propto r_{train}(i)$  described in Section 4.5.1 breaks up. The approach consists of splitting the data by picking a set  $T$  of candidate items, and a number  $\eta$  of test ratings per item so that  $|T|\eta/r = \sigma$ . For this to be possible, it is necessary that  $(1 - \varepsilon)r(i) \geq \eta, \forall i \in T$ , where  $\varepsilon$  is a minimum ratio of training ratings per item we consider appropriate. In particular, in order to allow for  $n$ -fold cross-validation, we should have  $\varepsilon \geq 1/n$ . The selection of  $T$  can be done in several ways. We propose to do so in a way that it maximises  $|T|$ , i.e., to use as many different target test items as possible, avoiding a biased selection towards popular items. If we sort  $i_k \in \mathcal{I}$  by popularity rank, it can be seen that this is achieved by picking  $T = \{i_k \in \mathcal{I} | k \leq \zeta\}$  with  $\zeta = \max \{k | (1 - \varepsilon)r(i_k)k/r \geq \sigma\}$ , so that  $\eta = (1 - \varepsilon)r(i_\zeta)$ . Figure 4.5b illustrates this procedure.

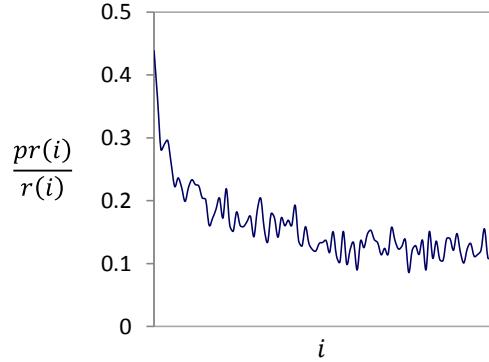
The expected effect of this approach is that the statistical relation  $E[pr_{test}(i)] \propto pr(i)$  no longer holds, and neither should hold now, as a consequence, the rationale described in Section 4.5.1 for popularity being the optimum non-personalised recommender. In fact, since  $E[pr_{test}(i)] = \eta \cdot pr(i)/r(i)$  for any  $i \in T$ , and  $\eta = \sigma \cdot r/|T|$ , it can be seen that if  $\mathcal{C} = T$  (TI policy) Equation (4.3) for AR yields:

$$E[P_s@n] = \frac{\sigma \cdot r}{n \cdot t |\mathcal{U}|} \sum_{k=1}^n \text{avg}_{u \in \mathcal{U}} \frac{pr(i_k^{u,s})}{r(i_k^{u,s})}$$

for any non-personalised recommender. If the ratio  $pr(i_k^{u,s})/r(i_k^{u,s})$  of positive ratings does not depend on  $k$ , we have  $E[P_s@n] = E[P_{RND}@n] = \sigma \cdot \delta$ . This means that popularity recommendation may get some advantage over other recommenders only if – and to the extent that – popular items have a higher ratio of positive ratings than unpopular items, and popularity recommendation will degrade to random precision otherwise. On the other hand, it can be seen that if  $\mathcal{C} \supsetneq T$  (i.e., the TI policy is not adhered to), then  $E[P_{RND}@n]$  would get reduced by a factor of  $|T|/|\mathcal{C}|$ .

For a non-personalised recommender in a 1R design, elaborating from Equations (4.2) and (4.3) we get:

$$\frac{r}{n \cdot t \cdot pr} \text{avg}_{u \in \mathcal{U}} \sum_{k=1}^n \frac{pr(i_k^{u,s})}{r(i_k^{u,s})} \leq E[1RP_s@n] \leq \frac{r}{n \cdot t \cdot pr} \text{avg}_{u \in \mathcal{U}} \sum_{k=1}^{m_u} \frac{pr(i_k^{u,s})}{r(i_k^{u,s})},$$



**Figure 4.6. Positive ratings ratio vs. popularity rank in MovieLens 1M. The graphic plots  $pr(i)/r(i)$ , where items are ordered by decreasing popularity. We display averaged values for 100 popularity segments, for a smoothed trend view.**

an equivalent situation where the measured precision of popularity recommendation is bound by the potential dependence between the ratio of positive ratings and popularity.

Figure 4.6 shows this ratio as  $pr(i)/r(i)$  with respect to the item popularity rank in MovieLens 1M. It can be seen that indeed the ratio grows with popularity in this dataset, which does lend an advantage for popularity recommendation. Even so, we may expect the bias to be moderate – but this has to be tested empirically, as it depends on the dataset. Note also that in applications where all ratings are positive (as e.g. in our Last.fm setup), popularity – and any non-personalised recommender – would drop exactly to random precision ( $E[P_s@n] = \sigma \cdot \delta$  in AR and  $1/t$  in 1R).

A limitation of this approach is that the formation of  $T$  may impose limits on the value of  $\sigma$ , and/or the size of  $T$ . If the popularity distribution is very steep,  $T$  may turn out small and therefore biased to a few popular items. Moreover, there is in general a solution for  $T$  only up to some value of  $\sigma$  – it is easy to see (formally, or just visually in Figure 4.5) that as  $\sigma \rightarrow 1$  there is no item for which  $(1 - \varepsilon) r(i_k) k/r \geq \sigma$ , unless the popularity distribution was uniform, which is never the case in practice. We have however not found these limitations to be problematic in practice, and common configurations turn out to be feasible without particular difficulty. For instance, in MovieLens 1M we get  $|T| = 1,703$  for  $\sigma = 0.2$  with  $\varepsilon = 0.2$  (allowing for a 5-fold cross-validation), resulting in  $\eta = 118$  test ratings per item.

This method can be used, as noted, in both the AR and 1R approaches. We shall refer to these combinations as **UAR** and **U1R** respectively, where ‘U’ stands for the “uniform” number of item test ratings. In U1R it is important to set  $C = T$  in order to sample non-relevant items within  $T$  (i.e.,  $N_u \subset T$ , for the TI policy). Otherwise, popularity would have a statistical advantage over other recommenders, as it would systematically rank irrelevant items in  $N_u - T$  below any relevant item in  $T$ , whereas

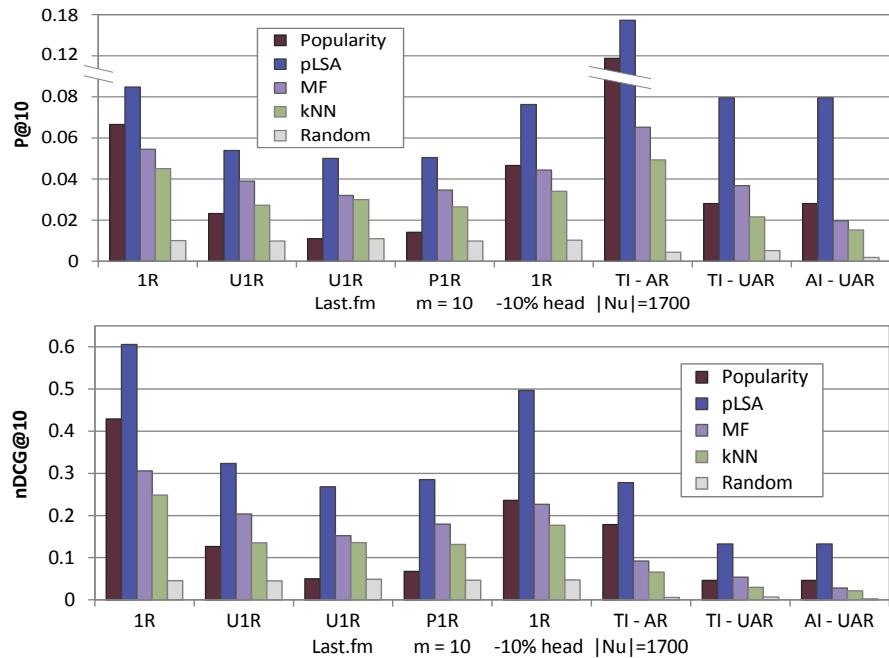
other algorithms might not. The same can be considered in UAR, unless the experimental setup requires  $|T_u| > |T|$ , as e.g. in the AI design. In that case a slight popularity bias would arise, as we shall see next.

### 4.6.3 Experimental Results

Figure 4.7 compares the results measured by 1R, AR and their corresponding popularity-neutralising variants. The setup is the same as in previous sections, except that for AR, we take TI-NN with  $|N_u| = 1,700$ , to level with UAR in random precision. All the results correspond to MovieLens 1M except Last.fm where indicated. It can be seen that P1R, U1R and UAR effectively limit the popularity bias. The techniques seem to be more effective on 1R than AR: U1R and (even more) P1R actually place the popularity algorithm by the level of random recommendation, whereas the measured popularity precision decreases in UAR, but remains above kNN. The advantage of popularity over randomness in U1R and P1R is explained by the bias in the ratio of positive ratings in popular items (Figure 4.6). This ratio is constant in Last.fm, whereby popularity drops to random in U1R, as predicted by our analysis in the previous section, proving that the popularity bias remaining in the uniform-test approach is caused by this factor. This residual bias is higher in U1R than P1R, because in the former,  $N_u$  is sampled over a larger popularity interval ( $|T| = 1,703$  vs.  $|T| / 10 = 370$  items), giving a higher range for advantage by popularity, which also explains why the latter still overcomes kNN in UAR. We may observe the importance of using the TI policy in UAR, without which (in AI-UAR) a higher bias remains. We also show the effect of removing the 10% most popular head items from the test data (and also from  $\mathcal{C}$ , i.e., they are excluded from  $N_u$  sampling) in 1R, as a simple strategy to reduce the popularity bias (Cremonesi et al., 2010). We see that this technique reduces the measured precision of popularity, but it is not quite as effective as the proposed approaches.

It is finally worth emphasising how **the percentile and uniform-test approaches discriminate between pure popularity-based recommendation and an algorithm like pLSA**, which does seem to take popularity as one of its signals, but not the only one. The proposed approaches allow uncovering the difference, neutralising popularity but not pLSA, which remains the best algorithm in all configurations.

As we mentioned in Section 4.3, we have taken precision as a simple and common metric for our study, but all the presented analysis and proposed alternatives straightforwardly generalise to other standard IR metrics, such as MAP, nDCG, and Mean Reciprocal Rank (MRR). Their application is direct in the AR setting; and they can be applied in 1R by simply introducing them in place of precision in the internal summation of Equation (4.1). Figure 4.7 shows results for nDCG, where we see that the analysed patterns hold just the same. The AR approach provides room for a



**Figure 4.7. Precision and nDCG of recommendation algorithms on MovieLens 1M (and Last.fm only where indicated) using the 1R, U1R, P1R ( $m = 10$  percentiles), AR, and UAR methodologies. The “-10% head” bars show the effect of removing the 10% most popular items from the test data (Cremonesi et al., 2010).**

slightly wider metric variety than 1R, in the sense that some metrics reduce to each other in 1R. For instance, for a single relevant item, MAP is equivalent to Mean Reciprocal Rank ( $MRR = 1/k$  where  $k$  is the rank of the first relevant item). And nDCG is insensitive to relevance grades in 1R (the grade of the single relevant item cancels out), whereas grades do make a difference in AR.

## 4.7 Conclusions

The application of Information Retrieval methodologies to the evaluation of recommender systems is not necessarily as straightforward as it may seem. Hence, it deserves close analysis and attention to the differences in the experimental conditions, and their implications on the explicit and implicit principles and assumptions on which the metrics build. We have proposed a systematic characterisation of design alternatives in the adaptation of the Cranfield paradigm to recommendation tasks, aiming to contribute to the convergence of evaluation approaches. We have identified assumptions and conditions underlying the Cranfield paradigm which are not granted in usual recommendation experiments. We have detected and examined resulting statistical biases, namely test sparsity and item popularity, which do not arise in common test collections from IR, but do interfere in recommendation experi-

ments. Sparsity is clearly a noisy variable that is meaningless with respect to the value of a recommendation. Whether popularity is in the same case is less obvious; we propose experimental approaches that neutralise this bias, leaving way to an unbiased observation of recommendation accuracy, isolated from this factor. With a view to their practical application, we have identified and described the pros and cons of the array of configuration alternatives and variants analysed in this study.

In general, we have found that evaluation metrics computed in AR and 1R approaches differ in how they are averaged. This means, more specifically, that precision obtained by approaches following a 1R design is bound linearly by precision of AR approaches. Moreover, we have observed that a percentile-based evaluation considerably reduces the margin for the popularity bias, although the main limitation of this approach is that it specifies a constraint on the size of the possible target sets. Additionally, a uniform-test approach removes any statistical advantage provided by having more positive test ratings. Furthermore, we have found that both approaches discriminate between pure popularity-based recommendation and an algorithm like pLSA.

The main goal of our research addresses a second-order problem: we aim to predict the accuracy of the predictions of recommendation algorithms. As we shall see, the (second-order) evaluation of our researched methods relies on the (first-order) evaluation metrics and methodologies by which the recommendation algorithms' accuracy is measured. In order to consistently evaluate our methods, the primary recommendation evaluation has to be reliable and well-understood. Any bias in the process would lead to inconclusive or misleading results about the predictive power of our methods. For this reason, the results presented in this chapter are a necessity for the main goal of this thesis, but the outcome can be of more general use. Specifically, in the following chapters we shall compare how the different methodologies (with and without neutralised biases) may impact the observations on the predictive power of our predictors.

The popularity effects in recommender systems have started to be reported in recent work (Cremonesi et al., 2011; Cremonesi et al., 2010; Steck, 2011). Our research complements such findings by seeking principled theoretical and empirical explanations for the biases, and providing solutions within the frame of IR evaluation metrics and methodology – complementarily to the potential definition of new special-purpose metrics (Steck, 2011). The extent to which popularity is a noisy signal may be further analysed by developing more complete metric schemes incorporating gain and cost dimensions, where popular items would expectably score lower. Such metrics may e.g. account for the benefits (to both recommendation consumers and providers) drawn from novel items in typical situations (Vargas and Castells, 2011), as a complement to plain accuracy. Online tests with real users should also be valuable for a comparative assessment of offline observations, and the validation of experimental alternatives.

## Part III

# Predicting performance in recommender systems

*You can never get a cup of tea large enough or  
a book long enough to suit me.*

Clive Staples Lewis



# Chapter 5

## Performance prediction in Information Retrieval

Information retrieval performance prediction has been mostly addressed as a query performance issue, which refers to the performance of an information retrieval system in response to a specific query. It also relates to the appropriateness of a query as an expression of the user's information needs. In general, performance prediction methods have been classified into two categories depending on the used data: pre-retrieval approaches, which make the prediction before the retrieval stage using query features, and post-retrieval approaches, which use the rankings produced by a retrieval engine. In particular, the so-called clarity score predictor – of special interest for this thesis – has been defined in terms of language models, and captures the ambiguity of a query with respect to the utilised document collection, or a specific result set.

In this chapter we provide an overview of terminology, techniques, and evaluation related to performance prediction in Information Retrieval. In Section 5.1 we introduce terminology and fundamental concepts of the performance prediction problem. In Section 5.2 we describe the different types of performance prediction approaches, which are mainly classified in the two categories mentioned above: pre-retrieval and post-retrieval approaches. Then, in Section 5.3 we provide a thorough analysis on the use of clarity score as a performance prediction technique, including examples, adaptations, and applications found in the literature. Finally, in Section 5.4 we introduce the general methodology used to evaluate performance predictors, along with the most common methods to measure their quality.

## 5.1 Introduction

Performance prediction has received little attention, if any, to date in the Recommender Systems field. Our research, however, finds a close and highly relevant reference in the adjacent Information Retrieval discipline, where performance prediction has gained increasing attention since the late 90's, and has become an established research topic in the field. Performance prediction finds additional motivation in personalised recommendation, inasmuch the applications they are integrated in may decide to produce recommendations or hold them back, delivering only the sufficiently reliable ones. Moreover, the ability to predict the effectiveness of individual algorithms can be envisioned as a strategy to optimise the combination of algorithms into ensemble recommenders, which currently dominate the field – rarely if ever are individual algorithms used alone in working applications, neither are they found individually in the top ranks of evaluation campaigns and competitions (Bennett and Lanning, 2007).

In Information Retrieval performance prediction has been mostly addressed as a query performance problem (Cronen-Townsend et al., 2002). Query performance refers to the performance of an information retrieval system in response to a particular query. It also relates to the appropriateness of a query as an expression of a user's information needs. Dealing effectively with poorly-performing queries is a crucial issue in Information Retrieval since it could improve the retrieval effectiveness significantly (Carmel and Yom-Tov, 2010).

In general, performance prediction techniques can be useful from different perspectives (Zhou and Croft, 2006; Yom-Tov et al., 2005a):

- From the user's perspective, it provides valuable feedback that can be used to direct a search, e.g. by rephrasing the query or suggesting alternative terms.
- From the system's perspective, it provides a means to address the problem of information retrieval consistency. The consistency of retrieval systems can be addressed by distinguishing poorly performing queries. A retrieval system may invoke different retrieval strategies depending on the query, e.g. by using query expansion or ranking functions based on the predicted difficulty of the query.
- From the system administrator's perspective, it may let identify queries related to a specific subject that are difficult for the search engine. According to such queries, the collection of documents could be extended to better answer insufficiently covered topics.
- From a distributed information retrieval's perspective, it can be used to decide which search engine (and/or database) to use, or how much weight give to different search engines when their results are combined.

Specifically, the performance prediction task in Information Retrieval is formalised based on the following three core concepts: **performance predictor**, **retrieval quality assessment**, and **predictor quality assessment**. In this context, the performance predictor is defined as a function that receives the query (and the result list  $D_q$  retrieved by the system, the set of relevant documents  $R_q$ , collection statistics  $\mathcal{C}$ , etc.), and returns a prediction of the retrieval quality for that query. Then, by means of a predictor quality assessment method, the predictive power of the performance predictor is estimated.

Based on the notation given in (Carmel and Yom-Tov, 2010), the problem of performance prediction consists of estimating a true retrieval quality metric  $\mu(q)$  (retrieval quality assessment) of an information retrieval system for a given query  $q$ . Hence, a performance predictor  $\hat{\mu}(q)$  has the following general form:

$$\hat{\mu}(q) \leftarrow \gamma(q, R_q, D_q, \mathcal{C}) \quad (5.1)$$

The prediction methods proposed in the literature establish different functions  $\gamma$ , and use a variety of available data, such as the query's terms, its properties with respect to the retrieval space (Cronen-Townsend et al., 2002), the output of the retrieval system – i.e.,  $D_q$  and  $R_q$  – (Carmel et al., 2006), and the output of other systems (Aslam and Pavlu, 2007).

According to whether or not the retrieval results are used in the prediction process, such methods can be classified into pre-retrieval and post-retrieval approaches, which are described in Sections 5.2.1 and 5.2.2, respectively. Another relevant distinction is based on whether the predictors are trained or not, but this classification is less popular, and will not be considered here.

Moreover, the standard methodology to measure the effectiveness of performance prediction techniques (that is, the predictor quality assessment method) consists of comparing the rankings of several queries based on their actual precision – in terms of a evaluation metric such as MAP – with the rankings of those queries based on their performance scores, i.e., their predicted precision. In Section 5.4 we detail this methodology, along with several techniques for comparing the above rankings.

### 5.1.1 Notion of performance in Information Retrieval

In order to identify good performance predictors, validating or assessing their potential, we first have to define metrics of actual performance. Performance metrics and evaluation have been a core research and standardisation area for decades in the Information Retrieval field. In this section we introduce and summarise the main performance metrics and evaluation methodologies developed in the field.

The notion of performance in general, and in Information Retrieval in particular, leads itself to different interpretations, views and definitions. A number of methods

for measuring performance have been proposed and adopted (Hauff et al., 2008a; Hauff, 2010), the most prominent of which will be summarised herein; see (Baeza-Yates and Ribeiro-Neto, 2011) for an extended discussion.

As a result of several decades of research by the Information Retrieval community, a set of standard performance metrics has been established as a consensual reference for evaluating the goodness of information retrieval systems. These metrics generally require a collection of documents and a query (or alternative forms of user input such as item ratings), and assume a ground truth notion of relevance – traditional notions consider this relevance as binary, while others, more recently proposed, consider different relevance degrees.

One of the simplest and widespread performance metrics in Information Retrieval is **precision**, which is defined as the ratio of retrieved documents that are relevant for a particular query. In principle, this definition takes all the retrieved documents into account, but can also consider a given cut-off rank as the **precision at n** or **P@n**, where just the top-n ranked documents are considered. Other related and widespread metric is **recall**, which is the fraction of relevant documents retrieved by the system. These two metrics are inversely related, since increasing one generally reduces the other. For this reason, usually, they are combined into a single metric – e.g. the **F-measure**, and the **Mean Average Precision** or **MAP** –, or the values of one metric are compared at a fixed value of the other metric – e.g. the **precision-recall curve**, which is a common representation that consists of plotting a curve of precision versus recall, usually based on 11 standard recall levels (from 0.0 to 1.0 at increments of 0.1).

An inherent problem of using MAP for poorly performing queries, and in general of any query-averaged metric, is that changes in the scores of better-performing queries mask changes in the scores of poorly performing queries (Voorhees, 2005b). For instance, the MAP of a baseline system in which the effectiveness is 0.02 for a query A, and 0.40 for a query B, is the same as the MAP of a system where query A doubles its effectiveness (0.04) and query B decreases a 5% (0.38). In this context, in (Voorhees, 2005a) two metrics were proposed to measure how well information retrieval systems avoid very poor results for individual queries: the **%no measure**, which is the percentage of queries that retrieved no relevant documents in the top 10 ranked results, and the **area measure**, which is the area under the curve produced by plotting MAP(X) versus X, where X ranges over the worst quarter queries. These metrics were shown to be unstable when evaluated in small sets of 50 queries (Voorhees, 2005b). A third metric was introduced in (Voorhees, 2006): **gmap**, the geometric mean of the average precision scores of the test set of queries. This metric emphasises poorly performing queries while it minimises differences between larger scores, remaining stable in small sets of queries (e.g. 50 queries) (Voorhees, 2005b).

Nonetheless, despite the above metrics and other efforts made to obtain better measures of query performance, MAP, and more specifically the **Average Precision per query**, are still widely used and accepted. See (Carmel et al., 2006; Cronen-Townsend et al., 2002; Hauff et al., 2008b; He and Ounis, 2004; He et al., 2008; Kompaoré et al., 2007; Zhao et al., 2008; Zhou and Croft, 2006; Zhou and Croft, 2007), among others.

Almost as important as the performance metric is the query type, which can be related to the different user information needs (Broder, 2002). Most work on performance prediction has focused on the traditional ad-hoc retrieval task where query performance is measured according to topical relevance (also known as content-based queries). Some work – such as (Plachouras et al., 2003) and (Zhou and Croft, 2007) – has also addressed other types of queries such as named page finding queries, i.e., queries focused on finding the most relevant web page assuming the queries contain some form of the “name” of the page being sought (Voorhees, 2002a).

When documents are timed (e.g. a newswire system), we can also distinguish two main types of queries that have been only partially exploited in the literature (Diaz and Jones, 2004; Jones and Diaz, 2007): those queries that favour very recent documents, and those queries for which there are more relevant documents within a specific period in the past.

Finally, we note that most of the research ascribed to predict performance has been focused not on predicting the “true” performance of a query (whatever that means), but on discriminating those queries where query expansion or relevance feedback algorithms have proved to be efficient from those where these algorithms fail, such as polysemic, ambiguous, and long queries. These are typically called *bad-to-expand* queries (Cronen-Townsend et al., 2006), illustrating the implicit dependence on their final application.

### 5.1.2 A taxonomy of performance prediction methods

Existing prediction approaches are typically categorised into pre-retrieval methods and post-retrieval methods (Carmel and Yom-Tov, 2010). Pre-retrieval methods make the prediction before the retrieval stage, and thus only exploit the query’s terms and statistics about these terms gathered at indexing time. In contrast, post-retrieval methods use the rankings produced by a search engine, and, more specifically, the score returned for each document along with statistics about such documents and their vocabulary.

| Category       | Sub-category   | Performance predictor (name and reference)  |
|----------------|----------------|---|
| Pre-retrieval  | Linguistics    | Morphological, syntactic, semantic:<br>(Mothe and Tanguy, 2005), (Kompaoré et al., 2007)  |
|                | Statistics     | Coherency:<br>coherence (He et al., 2008);<br>term variance (Zhao et al., 2008)<br>Similarity:<br>collection query similarity (Zhao et al., 2008)<br>Specificity:<br>IDF-based (Plachouras et al., 2004), (He and Ounis, 2004);<br>query scope (He and Ounis, 2004), (Macdonald et al., 2005);<br>simplified clarity: (He and Ounis, 2004)<br>Term relatedness:<br>mutual information (Hauff et al., 2008a)   |
| Post-retrieval | Clarity        | Clarity (Cronen-Townsend et al., 2002),<br>(Cronen-Townsend et al., 2006)<br>Improved clarity (Hauff, 2010) (Hauff et al., 2008b)<br>Jensen-Shannon Divergence (Carmel et al., 2006)<br>Query difficulty (Amati et al., 2004)   |
|                | Robustness     | Cohesion:<br>clustering tendency (Vinay et al., 2006);<br>spatial autocorrelation (Diaz, 2007);<br>similarity (Kwok et al., 2004), (Grivolla et al., 2005)<br>Document perturbation:<br>ranking robustness (Zhou and Croft, 2006);<br>document perturbation (Vinay et al., 2006)<br>Query perturbation:<br>query feedback (Zhou and Croft, 2007);<br>autocorrelation (Diaz and Jones, 2004) (Jones and Diaz, 2007);<br>query perturbation (Vinay et al., 2006);<br>sub-query overlap (Yom-Tov et al., 2005a)<br>Retrieval perturbation: (Aslam and Pavlu, 2007) |
|                | Score analysis | Normalised Query Commitment: (Shtok et al., 2009)<br>Standard deviation of scores: (Pérez-Iglesias and Araujo, 2009),<br>(Cummins et al., 2011)<br>Utility Estimation Framework: (Shtok et al., 2010)<br>Weighted Information Gain: (Zhou and Croft, 2007)  |

**Table 5.1. Overview of predictors presented in Section 5.2 categorised according to the taxonomy presented in (Carmel and Yom-Tov, 2010).**

**Pre-retrieval performance predictors** do not rely on the retrieved document set, but on other information mainly extracted from the query issued by the user, such as statistics computed at indexing time (e.g. inverse term document frequencies). They have the advantage that predictions can be produced before the system's response is even started to be elaborated, which means that predictions can be taken

into account to improve the retrieval process itself. However, they have a potential handicap with regards to their accuracy on the predictions, since extra retrieval effectiveness cues available with the system's response are not exploited (Zhou, 2007). Pre-retrieval query performance has been studied from two main perspectives: based on probabilistic methods (and more generally, on collection statistics), and based on linguistic approaches. Most research on the topic has followed the former approach. Some researchers have also explored inverse document frequency (IDF) and related features as predictors, along with other collection statistics.

**Post-retrieval performance predictors**, on the other hand, make use of the retrieved results. Broadly speaking, techniques in this category provide better prediction accuracy compared to pre-retrieval performance predictors. However, many of these techniques suffer from high computational costs. Besides, they cannot be used to improve the retrieval strategies without a post-processing step, as the output from the latter is needed to compute the predictions in the first place. In (Carmel and Yom-Tov, 2010) post-retrieval methods are classified as follows: 1) clarity based methods that measure the coherence (clarity) of the result set and its separability from the whole collection of documents; 2) robustness based methods that estimate the robustness of the result set under different types of perturbations; and 3) score analysis based methods that analyse the score distribution of results.

Table 5.1 shows a number of representative approaches on performance prediction, which will be described in the next section. These approaches are categorised according to the taxonomy and sub-categories proposed in (Carmel and Yom-Tov, 2010). In the table we can observe that the statistics category has been the most popular approach for pre-retrieval performance prediction. Several predictors have been categorised in the robustness category, probably due to its broad meaning (query, document, and retrieval perturbation). Finally, we note that recent effort from the community has been focused on the score analysis category.

## 5.2 Query performance predictors

In this section we explain the distinct performance predictors proposed in the literature. As mentioned before, based on whether or not retrieval results are needed to compute performance scores, predictors can be classified into two main types: pre-retrieval and post-retrieval predictors. In the following we summarise some of the approaches of each of the above types. For additional information, the reader is referred to (Carmel and Yom-Tov, 2010), (Hauff, 2010), and (Pérez Iglesias, 2012).

### 5.2.1 Pre-retrieval predictors

Pre-retrieval performance predictors do not rely on the retrieved document set, and exploit other collection statistics, such as the inverse document frequency (IDF). In this context, performance prediction has been studied from three main perspectives: based on linguistic methods, based on statistical methods, and based on probabilistic methods.

#### Linguistic methods

In (Mothe and Tanguy, 2005) and (Kompaoré et al., 2007) the authors consider 16 query features, and study their correlation with respect to average precision and recall. These features are classified into three different types according to the linguistic aspects they model:

- Morphological features:
  - **Number of words.**
  - **Average word length** in the query.
  - **Average number of morphemes per word**, obtained using the CELEX<sup>7</sup> morphological database. The limit of this method is the database coverage, which leaves rare, new, and misspelled words as mono-morphemic.
  - **Average number of suffixed tokens**, obtained using the most frequent suffixes from the CELEX database (testing if each lemma in a topic is eligible for a suffix from this list).
  - **Average number of proper nouns**, obtained by POS (part-of-speech) tagger's analysis.
  - **Average number of acronyms**, detected by pattern matching.
  - **Average number of numeral values**, also detected by pattern matching.
  - **Average number of unknown tokens**, marked by a POS tagger. Most unknown words happen to be constructed words such as “mainstreaming”, “postmenopausal” and “multilingualism.”
- Syntactic features:
  - **Average number of conjunctions**, detected through POS tagging.
  - **Average number of prepositions**, also detected through POS tagging.
  - **Average number of personal pronouns**, again detected through POS tagging.
  - **Average syntactic depth**, computed from the results of a syntactic analyser. It is a straightforward measure of syntactic complexity in terms of

---

<sup>7</sup> CELEX, English database (1993). Available at [www.mpi.nl/world/celex](http://www.mpi.nl/world/celex)

hierarchical depth; it simply corresponds to the maximum number of nested syntactic constituents in the query.

- **Average syntactic links span**, computed from the results of a syntactic analyser; it is the average pairwise distance (in terms of number of words) between individual syntactic links.
- Semantic features:
  - **Average polysemy value**, computed as the number of synsets in the WordNet<sup>8</sup> database that a word belongs to, and averaged over all terms of the query.

In the above papers the authors investigated the correlation between these features, and precision and recall over datasets with different properties, and found that the only feature that positively correlated with the two performance metrics was the number of proper nouns. Besides, many variables did not obtain significant correlations with respect to any performance metric.

### Statistical methods

Inverse document frequency is one of the most useful and widely used magnitudes in Information Retrieval. It is usually included in the information retrieval models to properly compensate how common terms are. Its formulation usually takes an ad hoc, heuristic form, even though formal definitions exist (Roelleke and Wang, 2008; Aizawa, 2003; Hiemstra, 1998). The main motivation for the inclusion of an IDF factor in a retrieval function is that terms that appear in many documents are not very useful for distinguishing a relevant document from a non-relevant one. In other words, it can be used as a measure of the specificity of terms (Jones, 1972), and thus as an indicator of their discriminatory power. In this way, IDF is commonly used as a factor in the weighting functions for terms in text documents. The general formula of IDF for a term  $t$  is the following:

$$\text{IDF}(t) = \log \frac{N}{N_t} \quad (5.2)$$

where  $N$  is the total number of documents in the system, and  $n_t$  is the number of documents in which the term  $t$  appears.

Some research work on performance prediction has studied IDF as a basis for defining predictors. He and Ounis (2004) propose a predictor based on the **standard deviation of the IDF** of the query terms. Plachouras et al. (2004) represent the quality of a query term by a modification of IDF where instead of the number of documents, the number of words in the whole collection is used (**inverse collection term**

---

<sup>8</sup> WordNet, lexical database for the English language. Available at <http://wordnet.princeton.edu/>

**frequency**, or ICTF), and the query length acts as a normalising factor. These IDF-based predictors displayed moderate correlation with query performance.

Other authors have taken the similarity of the query into account. Zhao et al. (2008) compute the vector-space based query similarity with respect to the collection, considered as a large document composed of concatenation of all the documents. Then, different **collection query similarity** predictors are defined based on the SCQ values (defined below) for each query term, by summing, averaging, or taking the maximum values:

$$\text{SCQ}(t) = (1 + \log \text{TF}(t)) \cdot \text{IDF}(t) \quad (5.3)$$

The similarity of the documents returned by the query has also been explored in the field. The inter-similarity of documents containing query terms is proposed in (He et al., 2008) as a measure of **coherence**, by using the cosine similarity between every pair of documents containing each term. Additionally, two predictors based on the **pointwise mutual information** (PMI) are proposed in (Hauff et al., 2008a). The PMI of two terms is computed as follows:

$$\text{PMI}(t_1, t_2) = \log \frac{p(t_1, t_2)}{p(t_1)p(t_2)} \quad (5.4)$$

where these probabilities can be approximated by maximum likelihood estimations, that is, based on collection statistics, where  $p(t_1, t_2)$  is proportional to the number of documents containing both terms, and  $p(t) \propto \text{TF}(t)$ . In that paper a first predictor is defined by computing the average PMI of every pair of terms in the query, whereas a second predictor is defined based on the maximum value. The predictive power of these techniques remains competitive, and is very efficient at run time.

## Probabilistic methods

These methods measure characteristics of the retrieval inputs to estimate performance. He and Ounis (2004) propose a **simplified** version of the **clarity score** (see next section) in which the query model is estimated by the term frequency in the query:

$$\text{SCS} = \sum_w P_{ml}(w|q) \log_2 \frac{P_{ml}(w|q)}{P(w|\mathcal{C})} \quad (5.5)$$

$$P_{ml}(w|q) = \frac{\text{qtf}}{\text{ql}}; P(w|\mathcal{C}) = \frac{\text{TF}(w)}{|V|}$$

where  $\text{qtf}$  is the number of occurrences of a query term  $w$  in the query,  $\text{ql}$  is the query length,  $\text{TF}(w)$  is the number of occurrences of a query term in the whole collection, and  $|V|$  is the total number of terms in the collection.

Despite its original formulation, where the clarity score can be considered as a pre-retrieval predictor (Cronen-Townsend et al., 2002), Cronen-Townsend and colleagues use result sets to improve the computation time. For this reason, it is typically classified as a post-retrieval predictor (Zhou, 2007; Hauff et al., 2008a), and thus, we describe it with more detail in the next sections.

Kwok et al. (2004) build a query predictor using support vector regression, by training classifiers with features such as document frequencies and query term frequencies. In the conducted experiments they obtained a small correlation between predicted and actual query performances. He and Ounis (2004) propose the notion of **query scope** as a measure of the specificity of a query, which is quantified as the percentage of documents that contain at least one query term in the collection, i.e.,  $\log(N_Q/N)$ , being  $N_Q$  the number of documents containing at least one of the query terms, and  $N$  the total number of documents in the collection. Query scope has shown to be effective in inferring query performance for short queries in ad hoc text retrieval, but very sensitive to the query length (Macdonald et al., 2005).

### 5.2.2 Post-retrieval predictors

Post-retrieval performance predictors make use of the retrieved results, in contrast to pre-retrieval predictions. Furthermore, computational efficiency is usually a problem for many of these techniques, which is balanced by better prediction accuracy. In the following we present the most representative approaches of each of the different sub-categories described in Section 5.1.2: clarity, robustness, and score analysis.

#### Clarity-based predictors

Cronen-Townsend et al. (2002) define **query clarity** as a degree of (the lack of) query ambiguity. Because of the particular importance and use of this predictor in the findings of this thesis, we shall devote a whole section (Section 5.3) for a thorough description and discussion about it. It is worth noting that the concept of query clarity has inspired a number of similar techniques. Amati et al. (2004) propose the **query difficulty** predictor to estimate query performance. In that work query performance is captured by the notion of the amount of information ( $Info_{DFR}$ ) gained after the ranking. If there is a significant divergence in the query-term frequencies before and after the retrieval, then it is assumed that the divergence is caused by a query that is easy to respond to.  $Info_{DFR}$  showed a significant correlation with average precision, but did not show any correlation between this predictor and the effectiveness of query expansion. The authors hence concluded that although the performance gains by query expansion in general increase as query difficulty decreases, very easy queries hurt the overall performance.

Adaptations of the query clarity predictor such as the one proposed in (Hauff et al., 2008b) will be discussed later in Section 5.3. Additionally, apart from the Kullback-Leibler divergence, the Jensen-Shannon Divergence on the retrieved document set and the collection also obtains a significant correlation between average precision and the distance measured (Carmel et al., 2006).

### **Robustness-based predictors**

More recently, a related concept has been coined: **ranking robustness** (Zhou and Croft, 2006). It refers to a property of a ranked list of documents that indicates how stable a ranking is in the presence of *uncertainty* in its documents. The idea of predicting retrieval performance by measuring ranking robustness is inspired by a general observation in noisy data retrieval. The observation is that the degree of ranking robustness against noise is positively correlated with retrieval performance. This is because the authors assumed that regular documents also contain *noise*, if noise is interpreted as uncertainty. The robustness score performs better than, or at least as well as, the clarity score.

Regarding document and query perturbation, Vinay et al. (2006) propose four metrics to capture the geometry of the top retrieved documents for prediction: the **clustering tendency** as measured by the Cox-Lewis statistic, the sensitivity to **document perturbation**, the sensitivity to **query perturbation**, and the **local intrinsic dimensionality**. The most effective metric was the sensitivity to document perturbation, which is similar to the robustness score. Document perturbation, however, did not perform well for short queries, for which prediction accuracy dropped considerably when alternative state-of-the-art retrieval techniques (such as BM25 or a language modelling approach) were used instead of the TF-IDF weighting (Zhou, 2007).

Several predictors have been defined based on the concept of query perturbation. Zhou and Croft (2007) propose two performance predictors are defined based on this concept specifically oriented for Web search. First, the **Weighted Information Gain** predictor measures the amount of information gained about the quality of retrieved results (in response to a query) from an imaginary state that only an average document (represented by the whole collection) is retrieved to a posterior state that the actual search results are observed. This predictor was very efficient and showed better accuracy than clarity scores. The second predictor proposed in that work is the **Query Feedback**, which measures the degree of corruption that results from transforming  $Q$  to  $L$  (the output of the channel when the retrieval system is seen as a noisy channel, i.e., the ranked list of documents returned by the system). The authors designed a decoder that can accurately translate  $L$  back into a new query  $Q'$ , whereupon the similarity between the original query  $Q$  and the new query  $Q'$  is taken as a performance predictor, since the authors interpreted the evaluation of the quality of

the channel as the problem of predicting retrieval effectiveness. The computation of this predictor requires a higher computational cost than the previous one, being a major drawback of this technique.

Additionally, in (Diaz and Jones, 2004) and (Jones and Diaz, 2007) the authors exploited **temporal features** (time stamps) of the document retrieved by the query. They found that although temporal features are not highly correlated to performance, using them together with clarity scores improves prediction accuracy. Similarly, Diaz (2007) proposes to use the spatial autocorrelation as a metric to measure spatial similarities between documents in an embedded space, by computing the Moran's coefficient over the normalised scores of the documents. This predictor obtained good correlations results, although the author explicitly avoided collections such as question-answering and novelty related under the hypothesis that documents with high topical similarity should have correlated scores and, thus, in those collections the predictor would not work properly.

Other predictor was proposed in (Jensen et al., 2005), where visual features such as document titles and snippets are used from a surrogate document representation of retrieved documents. Such predictor was trained on a regression model with manually labelled queries to predict precision at the top 10 documents in Web search. The authors reported moderate correlation with respect to precision.

In (Yom-Tov et al., 2005a) two additional performance predictors are proposed. The first predictor builds a **histogram of the overlaps** between the results of each sub-query that agree with the full query. The second predictor is similar to the first one, but is based on a decision tree (Duda et al., 2001), which again uses overlaps between each sub-query and the full query. The authors apply these predictors to selective query expansion detecting missing content, and distributed information retrieval, where a search engine has to merge ranks obtained from different datasets. Empirical results showed that the quality of the prediction strongly depends on the query length.

The following predictors have been based on the cohesion of the retrieved documents. Kwok et al. (2004) propose predicting query performance by analysing similarities among retrieved documents. The main hypothesis of this approach is that relevant documents are similar to each other. Thus, if relevant documents are retrieved at the top ranking positions, the similarity between top documents should be high. The preliminary results, however, were inconclusive since negligible correlations were obtained. A similar approach is proposed in (Grivolla et al., 2005), where the entropy and pairwise similarity among top results are investigated. First, the entropy of the set of the  $K$  top-ranked documents for a query was computed. In this case it was assumed that the entropy should be higher when the performance for a given query is bad. Second, the mean cosine similarity between documents was proposed, using the base form of TF-IDF term weighting to define the document vec-

tors. Correlation between average precision and the proposed predictors was not consistent along the different systems used in the experiment, although the predictors could still be useful for performance prediction, especially when used in combination.

### Predictors based on score analysis

Finally, the last family of post-performance predictors analyses the score distributions of the results for each query. We have to note that the Weighted Information Gain predictor (Zhou and Croft, 2007) explained above is sometimes categorised into this group. In the following we present other predictors where the retrieved scores are explicit in the predictor computation.

For instance, the **Normalised Query Commitment** (NQC) predictor (Shtok et al., 2009) measures the standard deviation of the retrieval scores, and applies a normalisation factor based on the score of the whole collection:

$$\text{NQC}(q) = \frac{\sqrt{1/|D_q| \sum_{d \in D_q} (s(d) - \mu_q)^2}}{|s(\mathcal{C})|} \quad (5.6)$$

where  $\mu_q$  is the mean score of results in  $D_q$  (the retrieved set of documents for a query  $q$ ). This predictor measures the divergence of results from their centroid, a “pseudo non-relevant document” that exhibits a relatively high query similarity (Carmel and Yom-Tov, 2010).

The **utility estimation framework** (UEF) was proposed in (Shtok et al., 2010) to estimate the utility of the retrieved ranking. In this framework three methods have to be specified to derive a predictor: a sampling technique for the document sets, a representativeness measure for relevance-model estimates, and a measure of similarity between ranked lists. Other authors have proposed approaches where standard deviation does not need to be computed for all the document scores in the retrieved results. Pérez-Iglesias and Araujo (2009) use a cutoff to decide how many documents are considered in the standard deviation computation. Moreover, Cummins et al. (2011) use different strategies to automatically select such cutoff.

Recently, Cummins (2012) has used Monte Carlo simulations to understand the correlations between average precision and the standard deviation of the scores in the head of a ranked list. The author found that the standard deviation of the list is positively correlated with the mean score of relevant documents, which in turn is positively correlated with average precision.

## 5.3 Clarity score

Cronen-Townsend et al. (2002) defined clarity score for Web retrieval as a measure of the lack of ambiguity of a particular query. More recently, it has been observed that this predictor also quantifies the diversity of the result list (Hummel et al., 2012). In this section we provide a deep analysis of this performance predictor since we shall use it along the rest of this thesis. We also describe examples and adaptations of the clarity score.

### 5.3.1 Definition of the clarity score

The clarity score predictor is defined as a Kullback-Leibler divergence between the query and the collection language model. It estimates the coherence of a collection with respect to a query  $q$  in the following way, given the vocabulary  $\mathcal{V}$  and a subset of the document collection  $R_q$  consisting of those documents that contain at least one query term:

$$\text{clarity}(q) = \sum_{w \in \mathcal{V}} p(w|q) \log_2 \frac{p(w|q)}{p(w|\mathcal{C})} \quad (5.7)$$

$$p(d|q) = p(q|d)p(d)$$

$$p(q|d) = \prod_{w_q \in q} p(w_q|d)$$

$$p(w|q) = \sum_{d \in R_q} p(w|d)p(d|q)$$

$$p(w|d) = \lambda p_{\text{ml}}(w|d) + (1 - \lambda)p_c(w)$$

The clarity value can thus be reduced to an estimation of the prior  $p(w|\mathcal{C})$  (collection language model), and the posterior  $p(w|q)$  of the query terms  $w$  (query language model) using  $p(w|d)$  over the documents  $d \in R_q$  and based on term frequencies and smoothing. It should be emphasised that if the set  $R_q$  is chosen as the whole collection  $\mathcal{C}$ , then this technique could be classified as a pre-retrieval performance predictor, since no information about the retrieval would be used. The importance of the size of the relevance set  $R_q$  (or number of feedback documents) has been studied in (Hauff et al., 2008b), where an adaptation of the predictor was proposed in order to automatically set the number of documents to consider.

As first published in (Cronen-Townsend et al., 2002) and (Cronen-Townsend et al., 2006), query ambiguity is defined as “the degree to which a query retrieves documents in the given collection with similar word usage.” Cronen-Townsend and

colleagues found that queries whose highly ranked documents are a mix of documents from disparate topics receive lower scores than if they result in a topically-coherent retrieved set, and reported a strong correlation between the clarity score and the performance of a query. Because of that, the clarity score method has been widely used in the area for query performance prediction.

Some applications and adaptations of the clarity score metric include query expansion (anticipating poorly performing queries that should not be expanded), improving performance in the link detection task (more specifically, in topic detection and tracking by modifying the measure of similarity of two documents) (Lavrenko et al., 2002), and document segmentation (Brants et al., 2002). More applications can be found in Section 5.3.3.

Zhou (2007) provides a complementary formulation of the clarity score by rewriting the formulation used above as follows:

$$\text{clarity}(q) = \sum_{w \in \mathcal{V}} \sum_{d \in R_q} p(w|d)p(d|q) \log \frac{\sum_{d \in R_q} p(w|d)p(d|q)}{p(w|\mathcal{C})} \quad (5.8)$$

In this way, Zhou emphasises, among other issues, the differences between the query clarity and the Weighted Information Gain predictor. Indeed, the author proposes the following generalisation of both formulations (for WIG and clarity). Specifically, the clarity formulation presented in Equations (5.7) and (5.8) is unified as follows:

$$\text{score}(q, \mathcal{C}, R) = \sum_{\xi \in T} \sum_{d \in R_q} \text{weight}(\xi, d) \log \frac{p(\xi, d)}{p(\xi, \mathcal{C})} \quad (5.9)$$

where  $T$  is a feature space, and  $R_q$  is a (ranked) document list. Besides this,  $d \in R_q \subseteq \mathcal{C}$  must be comparable somehow with elements  $\xi \in T$ , in order to make sensible functions  $\text{weight}(\xi, d)$  and  $p(\xi, d)$ . In this context, the query clarity as defined in (Cronen-Townsend et al., 2002) is an instantiation of Equation (5.9) where the following three aspects are considered:

- The feature space  $T$  is the whole vocabulary, consisting of single terms.
- The weight function is defined as  $\text{weight}(\xi, d) = p(w|d)p(d|q)$ .
- The function  $p(\xi, d)$  is defined as  $\sum_{d \in R_q} p(w|d)p(d|q)$ , that is, it uses a document model averaged over all documents in the ranked list.

These observations help to discriminate between the underlying models used by these two predictors. In particular, for the query clarity, they also contribute to capture not so obvious divergences between a query and the collection, as we shall see in the next section.

|              |                       |                               |
|--------------|-----------------------|-------------------------------|
| train (0.33) | train dog (0.65)      | obedience train dog (2.43)    |
|              |                       | railroad train dog (0.67)     |
|              | railroad train (0.73) | railroad train caboose (1.46) |

**Table 5.2. Examples of clarity scores for related queries.**

### 5.3.2 Interpreting clarity score in Information Retrieval

Aiming to better understand how the clarity score predictor behaves in Information Retrieval, and to what extent it is able to capture the difficulty or ambiguity of queries, in this section we summarise examples reported in the literature that let a clear interpretation of the predictor's values.

In a seminal paper (Cronen-Townsend et al., 2002) Cronen-Townsend and colleagues present the example shown in Table 5.2, which provides the clarity scores of a number of related queries that share some of their terms. These queries are related to each other in the sense that a particular query is formed by extending other query with an additional term, starting with an initial query formed by a single term, 'train' in the example. According to the queries of the table, we can observe that the term 'train' has different meanings for the largest queries; it refers to 'teach' in the query 'train dog', to the 'locomotive vehicle' in the query 'railroad train', and can refer to any of both meanings in the query 'railroad train dog.' The clarity scores capture the ambiguity of the queries (due to their different meanings for the term 'train'), independently from their length. In fact, the middle rightmost query 'railroad train dog' receives the lowest clarity score, corresponding to the most ambiguous query where the two considered meanings of 'train' are involved.

In the same paper, Cronen-Townsend and colleagues present the distribution of the language models for two queries, a clear query and a vague query (see Figure 2 in (Cronen-Townsend et al., 2002)). Each distribution is presented by plotting  $p(w|q) \log_2 p(w|q)/p(w|\mathcal{C})$  against the query terms  $w$ . The authors show that the distribution of the values of this function for the clear query dominates the distribution of the values of the vague query. This makes sense since the clarity score is computed by summing the probability values in the distribution of every term in the collection. Additionally, the authors show that the clear query presents spikes in its query language model when  $p(w|q)$  is plotted against the terms, and compared with the collection probability  $p(w|\mathcal{C})$ . Hence, some of the terms with high contribution from the query language model (i.e., with high  $p(w|q)$  values) obtain low collection

probabilities ( $p(w|\mathcal{C})$ ), thus evidencing a query that is different to the collection in its term usage (i.e., it is a non ambiguous query).

The above examples involve the (implicit) assumption known as *homogeneity assumption*, which specifies that the clarity score is higher if the documents in the considered collection are topically homogeneous. Hauff (2010) analyses the sensitivity of results with respect to that assumption. Specifically, the author computes the clarity score for three different ranked document lists: the relevant documents for a query, a non-relevant random sample, and a collection-wide random sample. The difference between the last two lists is that the second one is derived from documents judged as non-relevant, whereas the third one could contain any document in which at least one query term. Hauff shows how the clarity score is different depending on the origin of ranked document list, leading to a higher (lower) score by using relevant (non-relevant) documents for such list. However, we have to note that, as stated by Hauff, the quality in the separation of the clarity scores computed by each document list is different depending on the utilised dataset and queries.

The clarity score has been analysed in detail in Information Retrieval, mainly because its predictive power is superior to other performance predictors (in fact, it is one of the best performing post-retrieval predictors according to the overview presented in (Hauff, 2010)), but also because it provides interpretable results and high explanatory power in different IR processes, as we shall describe in the next section. Apart from that, the interest in this predictor is clear because of its probabilistic formulation and tight relationship with Language Models (Ponte and Croft, 1998).

### 5.3.3 Adaptations and applications of the clarity score

Cronen-Townsend and colleagues showed in (Cronen-Townsend et al., 2002) that clarity is correlated with performance, proving that the result quality is largely influenced by the amount of uncertainty involved in the inputs a system takes. In this sense, queries whose highly ranked documents belong to diverse topics receive lower scores than queries for which a topically-coherent result set is retrieved. Several authors have exploited the clarity score functionality and predictive capabilities (Buckley, 2004; Townsend et al., 2004; Dang et al., 2010), supporting its effectiveness in terms of performance prediction and high degree of adaptation. For instance, the predictor has been used for personalisation (Teevan et al., 2008) because of its proven capability of predicting ambiguity. In that paper the authors use more or less personalisation depending on the predicted ambiguity.

One of the first variants proposed in the area is the simplified clarity score proposed in (He and Ounis, 2004), presented in Section 5.2.1. In that paper He and Ouni changed the estimations of the posterior  $p(w|q)$  to simple maximum likelihood estimators. Hauff et al. (2008b) proposed the Improved Clarity – called Adapted Clarity in (Hauff, 2010) –, in which the number of feedback documents

$(R_q)$  is set automatically, and the term selection is made based on the frequency of the terms in the collection to minimise the contribution of terms with a high document frequency in the collection.

An alternative application of the clarity score is presented in (Allan and Raghavan, 2002), where the score obtained for the original set of documents returned by a query is compared against that obtained for a modified query, which was presumed to be more focused than the original one. Similarly, in (Buckley, 2004) Buckley uses the clarity score to measure the stability of the document rankings and compare it against a measure that uses the Mean Average Precision of each ranking (AnchorMap).

In (Sun and Bhownick, 2009), Sun and Bhownick adapted the concept of query clarity to image tagging, where a tag is visually representative if all the images annotated with that particular tag are visually similar to each other. In previous work (Sun and Datta, 2009) Sun and Datta proposed a similar concept, but in the context of blogging: a tag would receive a high clarity score if all blog posts annotated by the tag are topically cohesive.

Finally, an extension of the Kullback-Leibler divergence was proposed in (Aslam and Pavlu, 2007), where the Jensen-Shannon divergence was used instead. This distance is defined as the average of the Kullback-Leibler divergences of each distribution with respect to the average (or centroid) distribution. In this way, it is possible to compute the divergence between more than two distributions. Besides, the Jensen-Shannon divergence is symmetric, in contrast to the divergence used in the clarity score, and thus, a metric can be derived from it (Endres and Schindelin, 2003).

## 5.4 Evaluating performance predictors

In this section we describe the approaches proposed in the literature to evaluate the predictive power of a performance predictor. We define the different functions used to compute the quality of the performance predictors, most of them based on well known correlation coefficients between the true query performance values, and the expected or predicted performance values.

### 5.4.1 Task definition

Based on the notation presented in Section 5.1, in the following we present different techniques and functions to assess the effectiveness of performance predictors. Once the retrieval quality has been assessed ( $\mu(q)$ ), and the value of the performance predictor for each query is calculated ( $\hat{\mu}(q)$ , using the function  $\gamma$ ), the predictor quality is computed by using a predictor quality assessment function  $f^{qual}$  that measures the agreement between the true values of performance and the estimations, that is:

$$\text{Quality}(\gamma) = f^{qual}(\{\mu(q_1), \dots, \mu(q_n)\}, \{\hat{\mu}(q_1), \dots, \hat{\mu}(q_n)\}) \quad (5.10)$$

True quality values for each query are typically obtained by computing the per-query performance of a selected retrieval method (Cronen-Townsend et al., 2002; Hauff et al., 2008a), or by averaging the values obtained by several engines (Mothe and Tanguy, 2005), in order to avoid biases towards a particular method. As we shall see in the next section, the function  $f^{qual}$  typically represents a correlation coefficient; however, different possibilities are available and may be more appropriate depending on the prediction task.

In fact, in (Hauff et al., 2009) three estimation tasks were considered, by discriminating the output of the predictor function  $\hat{\mu}$ . **Query difficulty** estimation could be defined as a classification task where  $\hat{\mu} \rightarrow \{0,1\}$  indicates whether the query is estimated to perform well or poorly. The standard estimation of **query performance**, nonetheless, would be defined by a function  $\hat{\mu} \rightarrow \mathbb{R}$ , in order to provide a ranking of queries, where the highest score denotes the best performing query. Furthermore, as stated in (Hauff et al., 2009), this function by itself does not directly estimate the performance metric  $\mu$ . In order to do that we need to have normalised scores, such that the range of  $\hat{\mu}$  is compatible with that of the metric, which typically requires  $\hat{\mu} \rightarrow [0,1]$ . In this case, we would be considering the **normalised query performance** task.

The methodology described above is general enough to be applicable to any of these three tasks, but is clearly inspired by the second one, that is, the estimation of query performance and it can be easily applied also to third one (normalised performance prediction). Because of that, we describe next a recently proposed methodology more focused on the (binary) query classification task or query difficulty prediction described in (Pérez-Iglesias and Araujo, 2010).

Let us suppose that, instead of continuous values of the performance metric  $\mu$ , we are interested in estimating *as accurately as possible* the different difficulty grades of the queries, that is,  $\mu \rightarrow \{1, \dots, k\}$ , where  $k$  is the number of difficulty grades available. Obviously, the output of the predictor  $\hat{\mu}$  also has to be grouped in one of the  $k$  classes. Typically, we would have  $k = 3$ , representing “Easy”, “Average”, and “Hard” queries, although a binary partition could also be acceptable. In these terms the performance prediction problem is stated as a classification problem, where the goal is to effectively predict the query class.

Furthermore, this technique lets set, at the quality computation step, whether we want to weight uniformly each of the  $k$  classes, or if we are more interested in only one of them, by building, for instance, a confusion matrix, and applying standard Machine Learning evaluation metrics to a subset of it. In the next section we describe the most popular techniques for doing this, along with a new metric introduced in (Pérez-Iglesias and Araujo, 2010) oriented to the problem of performance prediction.

### 5.4.2 Measuring the quality of the predictors

There are several methods for measuring the quality of the performance prediction function  $\hat{\mu}$  defined in the previous section. In particular, the quality function  $f^{qual}$  may be able to capture linear relations, take into account the importance implied by the scores or the ordering given by each variable (true and estimated performance, i.e.,  $\mu$  and  $\hat{\mu}$ ), and exploit the implicit partitions derived by the method.

The most commonly used quality function is correlation, which has been measured by three well-known metrics: Pearson's, Spearman's, and Kendall's correlation coefficients. **Pearson's  $r$  correlation** captures linear dependencies between the variables, whereas **Spearman's  $\rho$**  and **Kendall's  $\tau$**  correlation coefficients are used in order to uncover non-linear relationships between the variables. They are generally computed as follows, although in special situations (in presence of ties, or when there are missing values in the data) alternative formulations may be used:

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (5.11)$$

$$\rho = 1 - \frac{6 \sum_{i=1}^n d(x_i, y_i)^2}{n(n^2 - 1)} \quad (5.12)$$

$$\tau = 1 - \frac{4Q(x, y)}{n(n - 1)} \quad (5.13)$$

where  $x$  and  $y$  represent the two variables of interest,  $\bar{x}$  and  $\bar{y}$  denote their means,  $d(x_i, y_i)$  is the difference in ranks between  $x_i$  and  $y_i$ , and  $Q(x, y)$  is the minimum number of swaps needed to convert the rank ordering of  $x$  to that of  $y$ . All these coefficients return values between  $-1$  and  $+1$ , where  $-1$  denotes a perfect anti-correlation,  $0$  denotes statistical independence, and  $+1$  denotes perfect correlation.

It can be observed that Spearman's  $\rho$  computes a Pearson's  $r$  between the ranks induced by the scores of the variables. Moreover, Kendall's  $\tau$  is the number of operations required to bring one list to the order of the other list using the *bubble sort* algorithm. Besides, although Spearman's and Kendall's correlations seem more general than Pearson's since they are able to capture non-parametric relations between the variables, we have to consider that distances between the scores are ignored in the rank-based coefficients, and thus, it is typically suggested to report one correlation coefficient of each type.

It is important to note that the number of points used to compute the correlation values affects the significance of the correlation results. The confidence test for a Pearson's  $r$  correlation, modeled as the  $t$ -value of a  $t$ -distribution (assuming normality) with  $N - 2$  degrees of freedom (being  $N$  the size of the sample), is defined by the following equation (Snedecor and Cochran, 1989):

| $p$ -value | N     |       |       | Pearson's $r$ value | N            |              |              |
|------------|-------|-------|-------|---------------------|--------------|--------------|--------------|
|            | 50    | 100   | 500   |                     | 50           | 100          | 500          |
| $p < 0.05$ | 1.677 | 1.661 | 1.648 | 0.1                 | 0.696        | 0.995        | <b>2.243</b> |
| $p < 0.01$ | 2.407 | 2.365 | 2.334 | 0.2                 | 1.414        | <b>2.021</b> | <u>4.555</u> |
|            |       |       |       | 0.3                 | <b>2.179</b> | <u>3.113</u> | <b>7.018</b> |
|            |       |       |       | 0.4                 | <u>3.024</u> | <b>4.320</b> | <u>9.739</u> |

**Table 5.3.** Left: minimum  $t$ -value for obtaining a significant value with different sample sizes (N). Right:  $t$ -value for a given Pearson's correlation value and N points. In bold when the correlation is significative for  $p < 0.05$ , and underlined for  $p < 0.01$ .

$$t = r \sqrt{\frac{N - 2}{1 - r^2}} \quad (5.14)$$

The  $t$ -value therefore depends on the size of the sample, and thus, the significance of a Pearson's correlation value  $r$  may change depending on the number of test queries. In particular, for small samples, we may eventually obtain strong but non-significant correlations; whereas for large samples, on the other hand, we may obtain significant differences, even though the strength of the correlation values may be lower. The above also applies to the correlations computed using the Spearman's coefficient, but only under the null hypothesis or large sample sizes (greater than 100) (Snedecor and Cochran, 1989; Zar, 1972). For Kendall's correlation, the confidence test can be computed using an exact algorithm when there are no ties based on a power series expansion in  $N^{-1}$ , depending again, thus, on the sample size (Best and Gipps, 1974).

Table 5.3 shows the minimum  $t$ -value for obtaining a significant value with different sample sizes and  $p$ -values, along with the  $t$ -value computed using Equation (5.14) for different correlation values and sample sizes. In the table we can observe that the same correlation value may be significant or not depending on the size of the sample, for instance, with 50 queries, observations are significant with  $p < 0.05$  for correlation values equal or above 0.3, whereas for 100 queries it is enough to obtain Pearson's correlation values of 0.2. This observation is related to the one presented in (Hauff et al., 2009), where Hauff and colleagues compared the confidence intervals of the three correlation coefficients described before, and observed how, due to the small query set sizes, most of the predictors analysed (pre-retrieval approaches such as clarity, IDF-based, and PMI) presented no significant differences, despite having very different values. In particular, this generated a subset of the analysed predictors that were not statistically different to the best performing predictor reported, and thus, any of the predictors in subset may be used in a later application since they obtain statistically similar (strictly speaking, not statistically different) correlations.

Furthermore, in the same paper, Hauff and colleagues proposed to use the **Root Mean Squared Error** (RMSE) as a quality function. The rationale behind this is that the RMSE squared is the function being minimised when performing a linear regression, and thus, it should also be able to capture the (linear) relation between the variables. In fact, there is a close relation between the RMSE and the Pearson's  $r$  coefficient, by means of the residual sum of squares (Carmel and Yom-Tov, 2010):

$$r^2 = 1 - \frac{SS_{err}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^n (x_i - y_i)^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \quad (5.15)$$

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^n (x_i - y_i)^2}{n - 1}} = \sqrt{\frac{SS_{err}}{n - 1}} \quad (5.16)$$

Additional extensions to these correlation coefficients have been proposed. Most of these extensions have been focused on incorporating weights in the computation of the correlation (Melucci, 2009; Yilmaz et al., 2008). However, despite these metrics have an evident potential in the performance prediction area, to the best of our knowledge there is no work using them in order to evaluate the quality of the predictors (Pérez Iglesias, 2012).

Finally, a different family of quality functions can be considered in the query difficulty task, that is, when the performance prediction is cast as a classification problem. These techniques are based on the accuracy of the classification provided by the performance predictor, and thus, classic Machine Learning techniques could be used. In (Pérez-Iglesias and Araujo, 2010), Pérez-Iglesias and Araujo propose to use the **F-measure**:

$$F = 2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \quad (5.17)$$

Additionally, in the same paper, Pérez-Iglesias and Araujo introduced a new metric (**distance based error measure**, or DBEM) along with a methodology that is focused on the misclassified difficulty classes between the predictor and the true classes. With this goal in mind, the authors apply a clustering algorithm to both the performance metric values and their estimations, aimed to minimise the distance between elements in the same group, and maximise the distance between elements in different groups. Specifically, Pérez-Iglesias and Araujo used the  $k$ -means algorithm, setting the value of  $k$  to the number of relevance grades,  $k = 3$  in their paper. The metric DBEM is defined as follows:

$$\text{DBEM} = \frac{\sum_i^n \text{dist}(c(x_i), c(y_i))}{\sum_i^n \max_j \text{dist}(c(x_i), c(x_j))} \quad (5.18)$$

$$\text{dist}(c_i, c_j) = \|i - j\|, 0 < i, j \leq k$$

where  $c(x)$  is the function which assigns the proper class or partition to a given score  $x$ , according to the clustering algorithm. This metric captures the distance between every partition, normalised by the maximum possible distance. In this case, lower distances imply a better predictor quality.

## 5.5 Summary

Improvement of the predictive capabilities to infer the performance or difficulty of a query is consolidated as a major research topic in Information Retrieval, where it has been mostly applied to ad-hoc retrieval. Several performance predictors have been defined based on many different information sources, demonstrating the usefulness of such predictors in different tasks, mainly for query expansion, but also for rank fusion, distributed information retrieval, and text segmentation.

Some issues are, however, still open in the field, mostly regarding the evaluation of performance prediction. Performance prediction methods have been usually evaluated on traditional TREC document collections, which typically consist of no more than one million relatively homogenous newswire articles, and few research work has exploited these techniques with larger datasets; see, e.g. (Carmel et al., 2006; Zhou, 2007; Hauff, 2010) for some exceptions. Furthermore, reported correlation coefficient values have been typically computed using a small number of points (e.g. 50 queries for standard tracks in TREC), not always providing enough confidence to derive conclusions. And more importantly, how predictors have to be evaluated and which metric has to be used are still open research questions, that have generated some fruitful discussion in recent publications (Hauff, 2010; Pérez Iglesias, 2012), although a definitive answer has not been obtained yet.

We may presume that in the future other information retrieval applications may benefit from the framework derived by these techniques, and may develop tailored performance predictors by using purpose-designed performance metrics and evaluation methodologies, such as the recently developed concept of document difficulty in (Alvarez et al., 2012). This thesis is an example of such an application in the Recommender Systems field. More specifically, as we shall see in the next chapter, we translate the problem of performance prediction to the Recommender Systems area, where it has been barely studied. We focus our research on the query clarity predictor as a basis for the recommendation performance predictors, although additional techniques could be used, as we shall also present in Chapter 6. Finally, among the array of evaluation strategies presented above, we have decided to use correlations since it is the most common one in the literature, and provides a fair notion about the interpretability of the results.

# Chapter 6

## Performance prediction in recommender systems

In this chapter, we state and address the recommendation performance prediction problem, proposing and evaluating different prediction schemes. After laying out a formal frame for the problem, we start by researching the adaptation of principles and prediction techniques that have been proposed and developed in ad-hoc Information Retrieval. More specifically, we draw from the notion of query clarity as a basis for finding suitable performance predictors that provide a well grounded theoretical formalisation. In analogy to query clarity, we hypothesise that the amount of uncertainty involved in user and item data (reflecting ambiguity in user's tastes and item popularity patterns) may also correlate with the accuracy of the system's recommendations. This uncertainty can be captured as the clarity of users and/or the clarity of items by an adaptation of the query clarity formulation. This adaptation, however, is not straightforward, as we shall describe. Besides the approaches elaborating on the notion of clarity, we propose new predictors based on theories and models from Information Theory and Social Graph Theory.

In Section 6.1 we formulate the research problem we aim to address. Next, in Sections 6.2, 6.3, and 6.4 we propose several performance predictors for recommender systems, some of them based on the clarity score, information theoretical related concepts – such as entropy –, and graph-based metrics. The proposed predictors are defined upon three different spaces, namely ratings, logs, and social networks. Moreover, we also provide specific correlations of the described predictors in Section 6.5 in order to show their predictive power under different conditions along with a discussion of the results. Finally, in Section 6.6 we provide some conclusions.

## 6.1 Research problem

Performance prediction finds a special motivation in recommender systems. Contrary to query-based information retrieval, as far as the initiative relies on the system, a performance prediction approach may provide a basis to decide producing recommendations or holding them back, depending on the expected level of performance on a per case basis, delivering only the sufficiently reliable cases. On the other hand, recommenders based on a single algorithm are not competitive in practice, and real applications heavily rely on hybridisations and ensembles of algorithms.

The capability to foresee which algorithm can perform better in different circumstances can therefore be envisioned as a good approach to enhance the performance of the combination of algorithms by dynamically adjusting the reliance on each subsystem. Furthermore, it is well-known in the recommender systems field that the performance of individual recommendation methods is highly sensitive to different conditions, such as data sparsity, quality and reliability, which are subject to an ample dynamic variability in real settings. Hence, being able to estimate in advance which recommenders are likely to provide the best output in a particular situation opens up an important window for performance enhancement. Alternatively, estimating which users of a system are likely to receive worse recommendations allows for modifications in the recommendation algorithms to predict this situation, and react in advance.

The problem of performance prediction has been however barely addressed in the Recommender Systems field. The issue has been nonetheless mentioned in the literature – evidencing the relevance of the problem – and is in some way often implicitly addressed by means of ad hoc heuristic tweaks such as significance weighting in nearest neighbour recommenders (Herlocker et al., 1999) and confidence scores (Wang et al., 2008a), along with additional computations (mainly normalisations) which are introduced into the recommendation methods aimed to better estimate the predicted ratings.

In the recommendation context, the problem of performance prediction can be stated as follows. We define a performance predictor as a function that takes a certain input, and returns a real value that correlates with some utility dimension of a recommender system. This is an instantiation of the problem presented in Section 5.1 but in the recommendation setting. For such purpose, we first specify more precisely what the input space of predictors consists of, and how the predictor input and output relate to the data involved in recommendation. Thus, a utility predictor handles the following information:

### Input variables

- The specific configuration of the recommender system. For instance, for a nearest neighbour recommender input parameters could be the neighbour map (that assigns a set of neighbours to each user) and a user similarity metric.
- Any input of the recommender, such as the active user and the active item.
- Background/context information: any known user, item, and user-item interaction data, such as user ratings, user features, item features, social network information, data timestamps, etc. We have to note that, even though the predictor will generally use this type information, we consider it as implicit input and do not include it explicitly in our notation to avoid making it needlessly cumbersome.

### Output variable

- A value in  $\mathbb{R}$ .

A predictor is thus a function  $\gamma: R \times \mathcal{U} \times \mathcal{I} \rightarrow \mathbb{R}$  ( $R$  being the set of all recommenders) that estimates the performance of the system, possibly using additional information available in the background. A predictor can be independent from some of these inputs, which would be then omitted in the previous notation. For instance, in this chapter we shall present predictors of the form  $\gamma: \mathcal{U} \rightarrow \mathbb{R}$  and  $\gamma: \mathcal{I} \rightarrow \mathbb{R}$ . Additionally, a predictor may assume a specific parameterised recommender algorithm family (e.g. nearest neighbour collaborative filtering), and needs some element of its configuration as input. It may also happen that a predictor does not make any assumption on the recommender – it does not depend on it – but still the predictor works well only for certain types of recommenders. It would be syntactically possible and correct to apply the predictor with other recommenders, although it may work badly. In general, what it means for a predictor to work “well” may depend on the application, but we generally assume it can be evaluated in terms of its correlation to some utility dimension of recommendations, such as an accuracy metric (RMSE, precision, nDCG) or alternative metrics such as novelty, diversity, etc.

If a recommender system can be decomposed into its internal configuration, then a predictor can directly take as input the components of the recommender configuration. For instance, neighbourhood-based collaborative filtering recommenders can be represented in  $R \equiv \mathcal{E} \times \mathcal{N} \times \mathcal{S}$ , where  $\mathcal{E}: \mathbb{R}^k \times \mathbb{R}^k \rightarrow \mathbb{R}^+$  is a preference estimation function (based on  $k$  similarity values between the target user and her neighbours, and  $k$  neighbours’ ratings on the target item),  $\mathcal{N}: \mathcal{U} \rightarrow P(\mathcal{U})$  is a neighbourhood assignment map, and  $\mathcal{S}$  is a similarity metric. Upon such a model, we would have  $\gamma: \mathcal{E} \times \mathcal{N} \times \mathcal{S} \times \mathcal{U} \times \mathcal{I} \rightarrow \mathbb{R}$ .

We may also constrain some inputs to a relevant condition they should meet. For instance, we could limit ourselves to a neighbourhood map that considers a user  $v$  as a candidate neighbour. In that case, this map can be essentially represented by  $v$ , and then we would have  $\gamma: \mathcal{E} \times \mathcal{U} \times \mathcal{S} \times \mathcal{U} \times \mathcal{I} \rightarrow \mathbb{R}$  (note that the first  $\mathcal{U}$  in the Cartesian product stands for neighbour users, and the second  $\mathcal{U}$  for target users).

It is important to note that when the predictor takes as input some of the inputs of the recommender, namely the active user and/or the active item, then the predictor's correlation with the recommender's utility must be measured on a per-input basis. For instance, if the predictor just takes users as input arguments, it should correlate with the average utility per user.

Moreover, predictors can also be used to enhance hybrid recommenders by favouring strategies that are predicted to produce better results. This can be done by relating activation switches in the recommenders to predictor values, so that one recommender or the others are activated or favoured depending on the predictor's estimation.

The way in which these activation switches are related to predictors is typically application-dependent. For instance, in ensemble recommenders consisting of a unique (Boolean) selection among a set of recommenders, the selection/discardng of recommenders can be a binary function of a predictor for each recommender. If the ensemble consists of a linear combination of recommenders, the weights in the combination can also be a function of the predictors. In neighbourhood-based collaborative filtering, activation switches can be the weights of neighbours in the prediction of user ratings. Indeed, relating predictor values to activation switches is a non-trivial problem and generally requires some research on itself.

Based on all the above mentioned issues, the general research problem we address consists of a) finding effective predictors of recommendation utility, and b) identifying and testing useful applications for the found predictors. In the remainder of this chapter we propose different predictors of recommendation utility using different types of input, namely ratings, logs, and social information. In Chapters 7 and 8 we shall exploit and evaluate such predictors in two applications: dynamic hybrid recommendation, and dynamic neighbour weighting in collaborative filtering.

## 6.2 Clarity for preference data: adaptations of query clarity

In this thesis, we propose different adaptations for the concept of query clarity to recommender systems. First, we deal with the definition of user clarity when rating-based preference data is available, where alternative ground models are proposed, depending on which random variables want to be considered in the computation of

the user clarity. Then, we define the concept of user clarity for log-based preference data. Additionally, for ratings we also define the concept of item clarity.

Now we propose a fairly general adaptation of query clarity, which may be instantiated into different schemes, depending on the input spaces considered. At an abstract level, we consider an adaptation that equates users in the recommendation domain to queries in the search domain, as the corresponding available representations of user needs in the respective domains. This adaptation results in the following formulation for **user clarity**:

$$\text{clarity}(u) = \sum_{x \in \mathcal{X}} p(x|u) \log_2 \frac{p(x|u)}{p(x)} \quad (6.1)$$

As we can observe, the clarity formulation strongly depends on a “vocabulary” space  $\mathcal{X}$ , which further constrains the user-conditioned model (or user model for short)  $p(x|u)$ , and the background probability  $p(x)$ . In ad-hoc information retrieval, this space is typically the space of words, and the query language model is a probability distribution over words (Cronen-Townsend et al., 2002). In recommender systems, however, we may have different interpretations, and thus, different formulations for such a probabilistic framework, as we shall show. In all cases, we will need to model and estimate two probability distributions: first, the probability that some event (depending on the current probability space  $\mathcal{X}$ ) is generated by the user language model (user model); and second, the prior probability of generating that event (background model).

Under this formulation, user clarity is in fact the difference (Kullback-Leibler divergence) between a user model and a background model. The use of user and background distributions as a basis to predict recommendation performance lies on the hypothesis that a user probability model being close to the background (or collection) model is a sign of ambiguity or vagueness in the evidence of user needs, since the generative probabilities for a particular user are difficult to single out from the model of the collection as a whole. In Information Retrieval, this fact is interpreted as a query for which the relevant documents are a mix of articles about different topics (Cronen-Townsend et al., 2002).

As an additional step, we generalise the adaptation stated in Equation (6.1) to allow for different reference probability models parameterised by a generic variable  $\theta$ .

$$\text{clarity}(u) = \mathbb{E}_\theta \left[ \sum_{x \in \mathcal{X}} p(x|u, \theta) \log_2 \frac{p(x|u, \theta)}{p(x|\theta)} \right] \quad (6.2)$$

This generalisation will allow for the development of further varieties of the clarity scheme, and simplifies to Equation (6.1) whenever we implicitly consider a fixed  $\theta$ , as we shall see next. Equivalently, the variable  $\theta$  may be integrated in both user and background models by exploiting a multidimensional vocabulary space:

$$\text{clarity}(u) = \sum_{x \in \mathcal{X}, \theta \in \Theta} p(x, \theta | u) \log_2 \frac{p(x, \theta | u)}{p(x, \theta)} \quad (6.2b)$$

It is easy to see that Equations (6.2) and (6.2b) are fully equivalent, and thus allow two interpretations for the same magnitude.

As stated in (Cronen-Townsend et al., 2002), language models capture statistical aspects of the generation of language. Therefore, if we use different vocabularies, we may capture different aspects of the user. The probabilistic relations between the variables involved in Equation (6.2) also depend on the nature of the data, and the different possible generative models induced by the recorded observations of user-item interactions (the input to a recommender system). In this thesis we consider two types of interaction data records: users-rating-items (where the atomic event is a user rating an item with a value), and users “consuming” items (a user accesses an item at some time instant). The first type fits a dataset such as MovieLens and CAMRa, and the second fits well Last.fm data – the datasets on which we shall test the methods to be developed here. Across these two types, in our research we explore mainly three vocabulary spaces for  $\mathcal{X}$ : ratings, items, and time. Each of the vocabulary spaces induces its own user-specific interpretation, as we shall see. As for the optional contextual parameter  $\theta$ , we shall consider here only the space of items ranging over the set of items – thus fully leveraging the triadic nature of the user-item-rating and user-item-time spaces. The scheme is however open to the exploration of further possibilities, as is the vocabulary space itself, beyond the options researched here.

In the following sections we thus explore several alternatives for rating-based and log-based data spaces (and their induced generative models).

### 6.2.1 Rating-based clarity

As just mentioned, in the rating space, we consider a set of user-item-rating tuples, where each user-item pair appears in a unique tuple (i.e., users only rate items once). We consider two possible vocabulary spaces: items and ratings, and two context alternatives: items (which make only sense in the rating vocabulary) and none. The resulting clarity schemes are summarised in Table 6.1, and have each their own interpretation.

The rating-based clarity model captures how differently a user uses rating values (regardless of the items the values are assigned to) with respect to the rest of users in the community. The item-based clarity takes into account which items have been rated by a user, and therefore, whether she rates (regardless of the rating value) the most rated items in the system or not. Finally, the item-and-rating-based clarity computes how likely a user would rate each item with some particular rating value, and compares that likelihood with the probability that the item is rated with some particular rating value. In this sense, the item-based user model makes the assumption that some items are more likely to be generated for some users than for others de-

| User clarity          | Vocabulary $\mathcal{X}$ / Context $\theta$ | User model  | Background model | Formulation  |
|-----------------------|---|-------------|------------------|--|
| Rating-based          | Ratings / None                              | $p(r u)$    | $p_c(r)$         | $\sum_r p(r u) \log_2 \frac{p(r u)}{p(r)}$                 |
| Item-based            | Items / None                                | $p(i u)$    | $p_c(i)$         | $\sum_i p(i u) \log_2 \frac{p(i u)}{p(i)}$                 |
| Item-and-rating-based | Ratings / Items                             | $p(r u, i)$ | $p_{ml}(r i)$    | $\sum_{r,i} p(i)p(r u, i) \log_2 \frac{p(r u, i)}{p(r i)}$ |

**Table 6.1. Three possible user clarity formulations, depending on the interpretation of the vocabulary and context spaces.**

pending on their previous preferences. The rating-based model, on the other hand, captures the likelihood of a particular rating value being assigned by a user, which is an event not as sparse as the previous one, with a larger number of observations. Finally, the item-and-rating-based model is a combination of the two previous models into a unified model incorporating items and ratings. As we mentioned before, this could be made more explicit by considering the user model  $p(r, i|u)$  in the Equation (6.2b), which would be equivalent to this model under some independence assumptions, i.e., when  $p(r, i|u) = p(r|u, i)p(i)$ .

### Ground models for user clarity

We ground the different clarity measures defined in the previous section upon a rating-oriented probabilistic model very similar to the approaches taken in (Hofmann, 2004) and (Wang et al., 2008a). The sample space for the model is the set  $\mathcal{U} \times \mathcal{I} \times \mathcal{R}$ , where  $\mathcal{U}$  stands for the set of all users,  $\mathcal{I}$  is the set of all items, and  $\mathcal{R}$  is the set of all possible rating values. Hence, an observation in this sample space consists of a user assigning a rating to an item. We consider three natural random variables in this space: the user, the item, and the rating value, involved in a rating assignment by a user to an item. This gives meaning to the distributions expressed in the different versions of clarity as defined in the previous section. For instance,  $p(r|i)$  represents the probability that a specific item  $i$  is rated with a value  $r$  – by a random user –,  $p(i)$  is the probability that an item is rated – with any value by any user –, and so on.

The probability distributions upon which the proposed clarity models are defined can use different estimation approaches, depending on the independence assumptions one would consider, and the amount of involved information. Background models are estimated using relative frequency estimators, that is:

$$p_c(r) = \frac{|\{(u, i) \in \mathcal{U} \times \mathcal{I} | r(u, i) = r\}|}{|\{(u, i) \in \mathcal{U} \times \mathcal{I} | r(u, i) \neq \emptyset\}|} \quad (6.3)$$

$$p_c(i) = \frac{|\{u \in \mathcal{U} | r(u, i) \neq \emptyset\}|}{|\{(u, j) \in \mathcal{U} \times \mathcal{I} | r(u, j) \neq \emptyset\}|}$$

$$p_{ml}(r|i) = \frac{|\{u \in \mathcal{U} | r(u, i) = r\}|}{|\{u \in \mathcal{U} | r(u, i) \neq \emptyset\}|}$$

$$p_{ml}(r|u) = \frac{|\{i \in \mathcal{I} | r(u, i) = r\}|}{|\{i \in \mathcal{I} | r(u, i) \neq \emptyset\}|}$$

These are maximum likelihood estimations in agreement with the meaning of the random variables as defined above. Starting from these estimations, user models can be reduced to the above terms by means of different probabilistic expansions and Bayesian reformulations, which we define next for the three models introduced in the previous section.

**Item based model.** The  $p(i|u)$  model can be simply expanded through marginalisation over ratings, but under two different assumptions: the item generated by the model only depends on the rating value, independently from the user or, on the contrary, depends on both the user and the rating. These alternatives lead to the following developments, respectively:

$$p_R(i|u) = \sum_{r \in \mathcal{R}} p_{ml}(i|r)p_{ml}(r|u) \quad (6.4)$$

$$p_{UR}(i|u) = \sum_{r \in \mathcal{R}} p(i|u, r)p_{ml}(r|u) \quad (6.5)$$

**Rating based model.** This model assumes that the rating value generated by the probability model depends on both the user and the item at hand. For this model, we sum over all possible items in the following way:

$$p(r|u) = \sum_{r(u,i)=r} p(r|u, i)p(i|u) \quad (6.6)$$

where the  $p(i|u)$  term can be developed as in the item-based model above. The term  $p(r|u, i)$  requires further development, which we define in the next model.

**Item-and-rating based model.** Three different models can be derived depending on how the Bayes' rule is applied. In these models, item probability is assumed to be uniform and thus it can be ignored in the computation of the expectation in Equation (6.2). In the same way as proposed in (Wang et al., 2008a), three relevance models can be defined, namely a user-based, an item-based, and a unified relevance model:

$$p_U(r|u, i) = \frac{p(u|r, i)p_{ml}(r|i)}{\sum_{r \in \mathcal{R}} p(u|r, i)p_{ml}(r|i)} \quad (6.7)$$

$$p_I(r|u, i) = \frac{p(i|u, r)p_{ml}(r|u)}{\sum_{r \in \mathcal{R}} p(i|u, r)p_{ml}(r|u)} \quad (6.8)$$

$$p_{UI}(r|u, i) = \frac{p(u, i|r)p_c(r)}{\sum_{r \in \mathcal{R}} p(u, i|r)p_c(r)} \quad (6.9)$$

The first derivation induces a user-based relevance model because it measures by  $p(u|r, i)$  how probable it is that a user rates item  $i$  with a value  $r$ . The item-based relevance model is factorised proportional to an item-based probability, i.e.,  $p_I(r|u, i) \propto p(i|u, r)$ . Finally, in the unified relevance model, we have  $p_{UI}(r|u, i) \propto p(u, i|r)$ . These estimations correspond respectively with the Equations 20a, 20b, and 21 from (Wang et al., 2008a); to make the thesis self-contained and facilitate the comparison between the different probability models, we present now these equations from (Wang et al., 2008a):

$$p(u|r, i) = \frac{1}{|S(r, i)|} \sum_{v \in S(r, i)} \frac{1}{h_u^{|J|}} K\left(\frac{\mathbf{u} - \mathbf{v}}{h_u}\right) \quad (6.10)$$

$$p(i|u, r) = \frac{1}{|S(r, u)|} \sum_{j \in S(r, u)} \frac{1}{h_i^{|U|}} K\left(\frac{\mathbf{i} - \mathbf{j}}{h_i}\right) \quad (6.11)$$

$$p(u, i|r) = \frac{1}{|S(r)|} \sum_{(v, j) \in S(r)} \frac{1}{h_u^{|J|}} K\left(\frac{\mathbf{u} - \mathbf{v}}{h_u}\right) \frac{1}{h_i^{|U|}} K\left(\frac{\mathbf{i} - \mathbf{j}}{h_i}\right) \quad (6.12)$$

where  $K(\cdot)$  is a Parzen Kernel function (Duda et al., 2001). In this formulation,  $\mathbf{u}$  denotes the user  $u$  represented as a vector by her ratings in the space of items. Unrated items can be filled with the average rating value or with other constant value, such as 0 or the average rating in the community. Respectively,  $\mathbf{i}$  represents the item  $i$  in the user space.  $h_u$  and  $h_i$  are the bandwidth window parameter for the user and item vector, respectively;  $S(\cdot)$  denotes the set of observed samples where event  $(\cdot)$  has happened. For example,  $S(r, i)$  denotes the set of observed samples with event  $(R = r, I = i)$ . More specifically:

$$S(r, i) = \{u \in \mathcal{U} | r(u, i) = r\} \quad (6.13)$$

$$S(r, u) = \{i \in \mathcal{I} | r(u, i) = r\} \quad (6.14)$$

$$S(r) = \{(u, i) \in \mathcal{U} \times \mathcal{I} | r(u, i) = r\} \quad (6.15)$$

In the experiments, we used a Gaussian Kernel function, i.e.,  $K(x) = e^{-x^2/2}/\sqrt{2\pi}$ , and  $h_i = h_u = 0.9$  as suggested in (Wang et al., 2008a).

| User clarity name | User dependent model       | Background model |
|-------------------|----------------------------|------------------|
| RatUser           | $p_U(r u, i); p_{UR}(i u)$ | $p_c(r)$         |
| RatItem           | $p_I(r u, i); p_{UR}(i u)$ | $p_c(r)$         |
| ItemSimple        | $p_R(i u)$                 | $p_c(i)$         |
| ItemUser          | $p_{UR}(i u)$              | $p_c(i)$         |
| IRUser            | $p_U(r u, i)$              | $p_{ml}(r i)$    |
| IRItem            | $p_I(r u, i)$              | $p_{ml}(r i)$    |
| IRUserItem        | $p_{UI}(r u, i)$           | $p_{ml}(r i)$    |

**Table 6.2. Different user clarity models implemented.**

Finally, different combinations of distribution formulations and estimations result in a fair array of alternatives. Among them, we focus on a subset that is shown in Table 6.2, which provide the most interesting combinations, in terms of experimental efficiency, of user and background distributions for each clarity model. These alternatives are further analysed in detail below (with examples) and in Section 6.5.1 where correlations obtained by each model are presented.

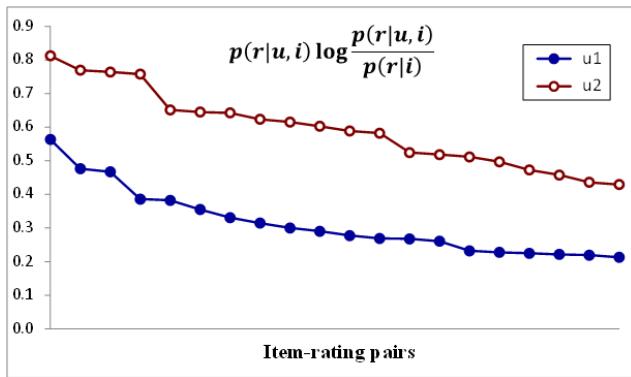
### Qualitative observation

In order to illustrate the proposed prediction framework and give an intuitive idea of what user characteristics the predictors are capturing, we show the relevant aspects of specific users that result in clearly different predictor values, in a similar way to the examples provided in (Cronen-Townsend et al., 2002) for query clarity. We compare three user clarity models out of the seven models presented in Table 6.2: one for each formulation included in Table 6.1. In order to avoid distracting biases on the clarity scores that a too different number of ratings between users might cause, we have selected pairs of users with a similar number of ratings. This effect would be equivalent to that found in Information Retrieval between the query length and its clarity for some datasets (Hauff, 2010).

Table 6.3 shows the details of two sample users on which we will illustrate the effect of the predictors. As we may see in the table,  $u_2$  has a higher clarity value than  $u_1$  for the three models analysed. That is, according to our theory,  $u_2$  is less “ambiguous” than  $u_1$ . Figure 6.1 shows the clarity contribution in a term-by-term basis for one of the item-and-rating-based clarity models – where, in this case, terms are equivalent to a pair (rating, item) – as analysed in (Cronen-Townsend et al., 2002). In the figure, we plot  $p(r|u, i) \log_2(p(r|u, i)/p(r|i))$  for the different terms in the collection, sorted in descending order of contribution to the user model, i.e.,

| User  | Number of ratings | ItemUser clarity | RatItem clarity | IRUserItem clarity |
|-------|-------------------|------------------|-----------------|--------------------|
| $u_1$ | 51                | 216.015          | 28.605          | 6.853              |
| $u_2$ | 52                | 243.325          | 43.629          | 13.551             |

**Table 6.3. Two example users, showing the number of ratings they have entered, and their performance prediction values for three user clarity models.**

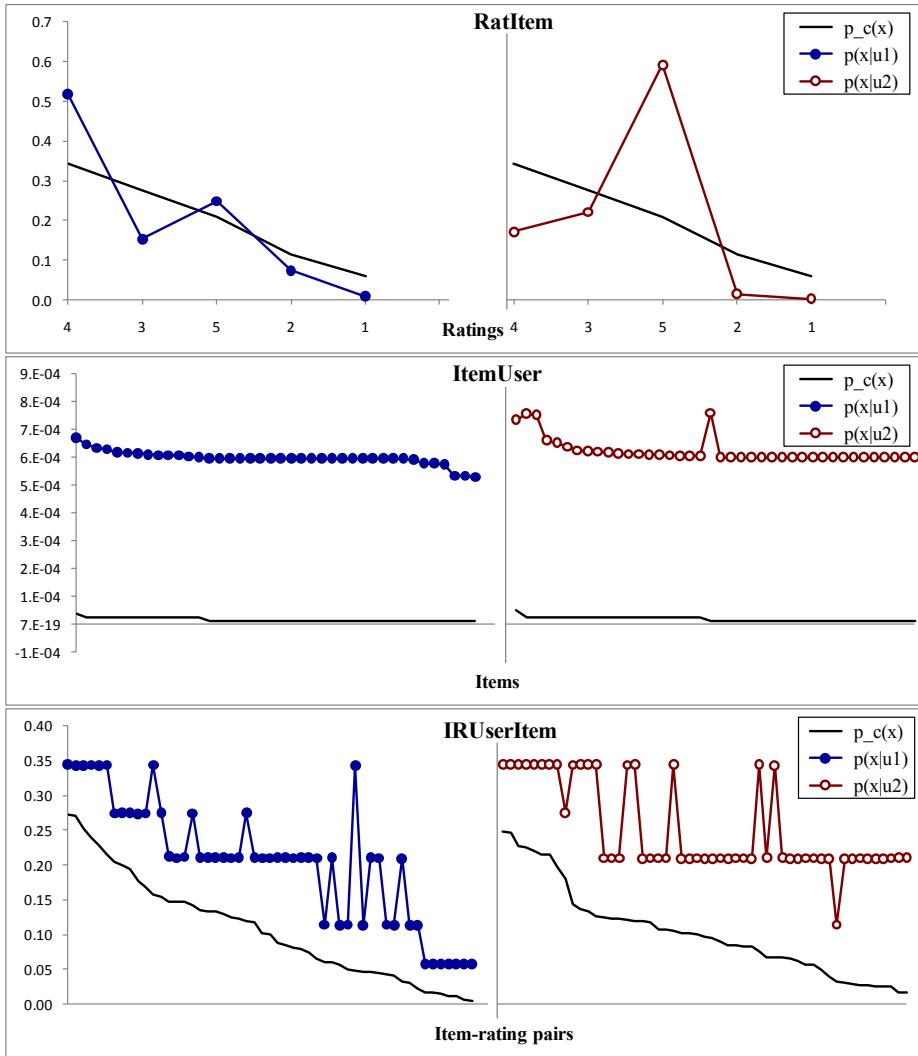


**Figure 6.1.** Term contributions for each user, ordered by their corresponding contribution to the user language model. IRUserItem clarity model.

$p(r|u, i)$ , for each user. For the sake of clarity, only the top 20 contributions are plotted. We may see how the user with the smaller clarity value receives lower contribution values than the other user. This observation is somewhat straightforward since the clarity value, as presented in Equation (6.1), is simply the sum of all these contributions, over the set of terms conforming the vocabulary. In fact, the figures are analogous for the rest of the models, since one user always obtains higher clarity value than the other.

Let us now analyse more detailed aspects in the statistical behaviour of the users that explain their difference in clarity. The IRUserItem clarity model captures how differently a user rates an item with respect to the community. Take for instance the top item-rating pairs for users 1 and 2 in the above graphic. The top pair for  $u_2$  is (4, “McHale’s Navy”). This means that the probability of  $u_2$  rating this movie with 4 is much higher than the background probability (considering the whole user community) of this rating for this movie. Indeed, we may see that  $u_2$  rated this movie with a 3, whereas the community mode rating is 1 – quite farther away from 4. This is the trend in a clear user. On the other extreme of the displayed values, the bottom term in the figure for  $u_1$  is (2, “Donnie Brasco”), which is rated by this user with a 5, and the community mode rating for this item is 4, thus showing a very similar trend between both. This is the characteristic trend of a non-clear user.

Furthermore, if we compare the background model with the user model, we obtain more insights about how our models are discriminating distinctive from mainstream behaviour. This is depicted in Figure 6.2. In this situation, we select those terms which maximise the difference between the user and background models. Then, for this subset of the terms, we sort the vocabulary with respect to its collection probability, and then we plot the user probability model for each of the terms in the vocabulary.



**Figure 6.2. User language model sorted by collection probability.**

These figures show how the most ambiguous user obtains a similar distribution to that of the background model, while the distribution of the less ambiguous user is more different. In the rating-based model this effect is clear, since the likelihood of not so popular rating values (i.e., a ‘5’) is larger for  $u_2$  than for  $u_1$ , and at the same time, the most popular rating value (a ‘4’) is much more likely for  $u_1$ . The figure about the ItemUser model is less clear in this aspect, although two big spikes are observed for  $u_1$  with respect to the collection distribution, which correspond with two unpopular movies: ‘Waiting for Guffman’ and ‘Cry, the beloved country’, both with a very low collection probability. Finally, the figure about the IRUserItem model successfully shows how  $u_2$  has more spikes than  $u_1$ , indicating a clear divergence from the background model; in fact,  $u_1$ ’s distribution partially mimics that of the collection. In summary, the different models proposed are able to successfully sepa-

| Item clarity          | Vocabulary $\mathcal{X}$ / Context $\theta$ | Item model | Background model | Formulation  |
|-----------------------|---|------------|------------------|--|
| Rating-based          | Ratings / None                              | $p(r i)$   | $p_c(r)$         | $\sum_r p(r i) \log_2 \frac{p(r i)}{p(r)}$               |
| User-based            | Users / None                                | $p(u i)$   | $p_c(u)$         | $\sum_u p(u i) \log_2 \frac{p(u i)}{p(u)}$               |
| User-and-rating-based | Ratings / Users                             | $p(r u,i)$ | $p_{ml}(r u)$    | $\sum_{r,u} p(u)p(r i,u) \log_2 \frac{p(r i,u)}{p(r u)}$ |

**Table 6.4. Three possible item clarity formulations, depending on the interpretation of the vocabulary and context spaces.**

rate information concerning the user and that from the collection, in order to infer whether a user is different or similar from the collection as a whole.

### Item clarity

Alternatively to user-based predictors, we can also consider item-based predictors, where the performance prediction is made on an item-basis. Item predictors can be defined analogously as those defined previously for users, the equation for **item clarity** being as follows:

$$\text{clarity}(i) = \mathbb{E}_\theta \left[ \sum_{x \in \mathcal{X}} p(x|i, \theta) \log_2 \frac{p(x|i, \theta)}{p(x|\theta)} \right] \quad (6.16)$$

The formulation of the item predictors we propose is basically equivalent to the user-based scheme but swapping users and items. That is, we have the three formulations presented in Table 6.4 where the vocabulary now may be either ratings or users, and the context variable is the user space. Based on these three formulations, and on derivations analogous to those presented before, we propose the seven item predictors defined in Table 6.5 which are further evaluated in Section 6.5.2.

In some of the instantiations of the item clarity predictor, we may observe that there are item probability models statistically equivalent to some of the user probability models, such as the  $p_U(r|i, u)$  and  $p_U(r|u, i)$ . For this reason, we now only spec-

| Item clarity name | Item dependent model       | Background model |
|-------------------|----------------------------|------------------|
| RatItem           | $p_I(r i, u); p_{IR}(u i)$ | $p_c(r)$         |
| RatUser           | $p_U(r i, u); p_{IR}(u i)$ | $p_c(r)$         |
| UserSimple        | $p_R(u i)$                 | $p_c(u)$         |
| UserID            | $p_{IR}(u i)$              | $p_c(u)$         |
| URItem            | $p_I(r i, u)$              | $p_{ml}(r u)$    |
| URUser            | $p_U(r i, u)$              | $p_{ml}(r u)$    |
| URItemUser        | $p_{IU}(r i, u)$           | $p_{ml}(r u)$    |

**Table 6.5. Different item clarity models implemented.**

ify those probability models which have not been defined before, for the rest of estimations see Equation (6.3):

$$p_R(u|i) = \sum_{r \in \mathcal{R}} p_{ml}(u|r)p_{ml}(r|i) \quad (6.17)$$

$$p_{IR}(u|i) = \sum_{r \in \mathcal{R}} p(u|i, r)p_{ml}(r|i) \quad (6.18)$$

$$p_{IU}(r|i, u) = p_{UI}(r|i, u) \quad (6.19)$$

## 6.2.2 Log-based clarity

In this section we adapt some of the previous models proposed for user clarity when the preference data come in the form of user-item interaction logs. Log data has a particularity we aim to exploit: the number of times a user consumes (purchased, listened, browsed, etc.) an item may be higher than one, in contrast with rating-based preferences, where the relation between a user and an item is summarised as a unique value, the rating. Moreover, the timestamp of the interactions has a stronger meaning in the implicit approach, as it informs of the very instant the user decided to use the item, rather than the time when the user decided to reflect on her quality of experience with the item (rating time). Specialised recommendation algorithms have been proposed in the literature that exploit such features in order to obtain better recommendations (Xiang et al., 2010; Lee et al., 2008). Additional alternatives for the definition of the vocabulary may be proposed, but we shall focus on these two: log co-occurrences and timestamps.

Specifically, based on Equation (6.2) and the three instantiations of  $\mathcal{X}$  and  $\theta$  shown in Table 6.1, in principle only an instantiation analogous to the second one ( $\mathcal{X} = \mathcal{I}$ , no context – to which we shall refer as frequency-based clarity) makes sense here, as there is no rating space. However, it is possible to consider an additional space, which leads to structurally similar instantiations by taking time as the  $\mathcal{X}$  vocabulary. The similarity is only syntactic, as the meaning and implications of the resulting magnitude, to which we shall refer as time-based clarity, are quite different from rating-based clarity – in other words, ratings and time are quite different dimensions –, as we shall describe later below.

### Frequency-based clarity

As mentioned above, we may define the following instantiation of the Equation (6.2) based on frequencies as follows:

$$\text{frequency-based clarity}(u) = \sum_i p(i|u) \log_2 \frac{p(i|u)}{p(i)} \quad (6.20)$$

where now the estimations of the user and background models are computed using directly the frequencies of the co-occurrences of some particular user-item interaction in the data:

$$\begin{aligned} p(i) &= \frac{\text{freq}(i)}{\sum_{j \in \mathcal{I}} \text{freq}(j)} \\ p(i|u) &= \frac{\text{freq}(i, u)}{\sum_{j \in \mathcal{I}_u} \text{freq}(j, u)} \end{aligned} \quad (6.21)$$

An alternative to such estimations is to use transformations from implicit log-based to explicit ratings, such as the one proposed in (Celma, 2008). In that approach, any of the predictors based on ratings proposed in the previous section could be applied, since these transformations give the additional vocabulary space of ratings that was absent in principle in log data.

### Time-based clarity

As introduced earlier, the second dimension susceptible to be exploited when log-based preference data are available is time. The time dimension is being paid increasing attention in Information Retrieval, where, for instance, it has been integrated into language models as a means to capture some temporal information needs from the user (Berberich et al., 2010), and the temporal query dynamics are being increasingly considered in the field (Kulkarni et al., 2011). In fact, temporal query features have also been used for query performance prediction, showing low or moderate correlation with query performance by themselves, although higher correlation is obtained when such features are combined with query clarity (Díaz, 2007; Díaz and Jones, 2004).

Furthermore, time has an inherent place in recommendation: recommender systems take as input (potentially long) histories of user interaction with items (Lathia, 2010; Zimdars et al., 2001; Burke, 2010). Time is an essential dimension in making sense of the data, and in explaining, analysing and interpreting the motivations behind the actions of users recorded over time. We propose to bring these ideas to recommender systems, in particular, to adapt the temporal features studied by Díaz and colleagues on a recommender system dataset. More specifically, we use the temporal Kullback-Leibler divergence described in (Díaz and Jones, 2004) as a starting point, which we generalise and elaborate upon by considering the instantiation of Equation (6.2) for a time-based space  $\mathcal{X}$ , and the space of items as a possible contextual dimension, as presented in Table 6.6. In the following, we define the specific instantiations of the temporal clarity formulations presented in this table.

| User clarity        | Vocabulary $\mathcal{X}$ / Context $\theta$ | User model  | Background model | Formulation  |
|---------------------|---|-------------|------------------|--|
| Time-based          | Time / None                                 | $p(t u)$    | $p(t)$           | $\sum_t p(t u) \log_2 \frac{p(t u)}{p(t)}$                 |
| Item-and-time-based | Time / Items                                | $p(t u, i)$ | $p(t i)$         | $\sum_{t,i} p(i)p(t u, i) \log_2 \frac{p(t u, i)}{p(t i)}$ |

**Table 6.6. Two temporal user clarity formulations, depending on the interpretation of the vocabulary space.**

**Time based model.** We denote as **TimeSimple clarity** the most direct adaptation for temporal clarity, which does not use any further extension over other dimensions. It simply computes  $p(t|u)$  using smoothing (see below) and  $p_c(t)$  from the collection frequencies.

**Item-and-time based model.** Like in the previous section, we develop conditional probabilities into sums with respect to a third variable: the items rated by the user. Here, we define two temporal clarity predictors depending on the distribution assumed for the items in the summation. If the distribution is uniform we denote such predictor as **ItemTime clarity** and  $p(i) = 1/|\mathcal{I}|$ . If, on the other hand, we also want to incorporate the popularity of the item for – which we have more data in this context and makes more sense than in rating data, since there the interaction between a user and an item is binary –, we include the prior item probability as  $p(i) = p_c(i)$ , which can be estimated considering the frequency by which  $i$  is accessed based on the interaction log.

The probabilities presented above are estimated as follows:

$$\begin{aligned}
p_c(t) &= \frac{|\{(v, j, t) \in \mathcal{L} | v \in \mathcal{U}, j \in \mathcal{J}\}|}{|\mathcal{L}|} \\
p_c(i) &= \frac{|\{(v, i, s) \in \mathcal{L} | v \in \mathcal{U}, s \in \mathcal{S}\}|}{|\mathcal{L}|} \\
p_{ml}(t|i) &= \frac{|\{(v, i, t) \in \mathcal{L} | v \in \mathcal{U}\}|}{|\{(v, i, s) \in \mathcal{L} | v \in \mathcal{U}, s \in \mathcal{S}\}|} \\
p_{ml}(t|u) &= \frac{|\{(u, j, t) \in \mathcal{L} | j \in \mathcal{J}\}|}{|\{(u, j, s) \in \mathcal{L} | j \in \mathcal{J}, s \in \mathcal{S}\}|} \\
p_{ml}(t|u, i) &= \frac{|\{(u, i, t) \in \mathcal{L}\}|}{|\{(u, i, s) \in \mathcal{L} | s \in \mathcal{S}\}|}
\end{aligned} \tag{6.22}$$

Note that the variable  $t$  in  $(u, i, t)$  in the above expressions denotes a timestamp in the discretised time segment (e.g. day, week) represented by  $t$ . Furthermore, these are simple estimations of the distributions; hence, it is also possible to introduce non-parametric estimations or additional expansions through similar users or items (Wang et al., 2006a; Wang et al., 2008a). Moreover, distributions can also be modeled by

other statistical theories or hypothesis (such as Bayesian inversion), and distribution fitting/modelling from time series theory could also be studied (Diaz and Jones, 2004; Wang et al., 2008b).

In particular, we have smoothed these estimations using Jelinek-Mercer as follows:

$$\begin{aligned} p(t|i) &= \lambda p_{ml}(t|i) + (1 - \lambda) p_c(t) \\ p(t|u) &= \lambda p_{ml}(t|u) + (1 - \lambda) p_c(t) \\ p(t|u, i) &= \lambda p_{ml}(t|u, i) + (1 - \lambda) p_c(t) \end{aligned} \quad (6.23)$$

### 6.3 Predictors based on social topology

Social information is widespread nowadays. As we surveyed in Chapter 2, recommender systems that use social information are proliferating in the research literature, as well as in the recommender system industry, because of the effectiveness they are being found to have. It seems therefore sensible to consider social information as a potentially useful input for predicting the performance of recommendation. The motivation for this approach is obvious when applied to social recommender systems, though we will also explore its potential properties in relation to non-explicitly social recommendation, in order to study whether social topologies may have an indirect effect on the results of the algorithms for different users.

With this goal in mind, we explore the use of graph-based measures as indicators of the user strength in the social network, which may in turn correlate with the ease or difficulty of users as recommendation targets. Graph-based measures developed from link-analysis theory are straightforward to interpret where they are often used to understand the structure of communities within a population (De Choudhury et al., 2010; Albert and Barabási, 2002). As a basis for user performance prediction they may thus bring an advantage in terms of explaining the predictions.

More specifically, the utilised indicators of the user strength in the network are based on the following vertex measures computed over the social network for each user, where a user is represented as a node in the graph, and the user's friends correspond with the node's neighbours:

- **Average neighbour degree:** mean number of friends of each user's friend (Kossinets and Watts, 2006).
- **Betweenness centrality:** indicator of whether a user can reach others on relative short paths (Freeman, 1977).
- **Clustering coefficient:** probability that the user's friends are friends themselves (Watts and Strogatz, 1998).

- **Degree:** number of the user’s friends in the social network (Milgram, 1967).
- **Ego components size:** number of connected components remaining when the user and her friends are removed (Newman, 2003).
- **HITS:** Kleinberg, 1999) defines two complementary measures which assign recursively a weight to each vertex (user) depending on the topology of the network. In this way, they define hubs and authorities: a vertex is a hub when it links to authoritative vertices, and is an authority when it links to hub vertices. Since the social network used here (see Appendix A.1.3) is undirected, hub and authority scores are redundant and we only report one, denoted as **HITS**.
- **PageRank score:** well-known measure of connectivity relevance within a social network based on a random walk over the vertices, where a probability ( $\alpha = 0.2$  in our experiments) of jumping to any other vertex is introduced (Brin and Page, 1998).
- **Two-hop neighbourhood size:** count of all the user’s friends plus all the user’s friend’s friends (De Choudhury et al., 2010).

## 6.4 Other approaches

As a reference for comparison, we shall also test further predictors besides the ones proposed in the thesis, directly drawn from the literature, and not necessarily based on probabilistic formalisations, but following more loose formalisations, or heuristic approaches. As a further sanity check, we shall also examine obvious and simple functions (such as the amount of activity of a user), as a reference for the justification of elaborate approaches as proposed. Next, we present these predictors which are evaluated and compared in Section 6.5.

### 6.4.1 Using rating-based preference data

A fairly simple user predictor against which we would like to compare more elaborate functions is the **count** predictor, namely the number of items a user has rated at some specific moment. This predictor, as we shall see later, can be defined in the training set and in the test set, and although its rationale is the same, the output has different implications. Whereas in training this predictor is measuring how much information a recommender knows about some specific user, in test this value would be related to the amount of relevance used to obtain the performance metric. Furthermore, as observed in Chapter 4, the amount of relevance would be different depending on the evaluation methodology considered. However, we have to note that, due to statistical effects, the training count (profile size in training) and test count

(profile size in test) would probably be related if the training/test split is performed randomly.

$$\text{count}(u) = |\mathcal{I}_u| = |\{(u, \cdot, \cdot)\}| \quad (6.24)$$

Two additional heuristic predictors can be defined by looking at user statistics such as the **mean** and the **standard deviation** of the user's ratings. It seems plausible that such predictors would not be equally powerful for any type of recommender: it would depend on whether these statistics are used by the recommender. For instance, one might have the intuition that the higher the standard deviation, the lower the recommendation performance as one may figure out uniform user ratings to be a somewhat easier target.

$$\text{mean}(u) = \frac{1}{|\mathcal{I}_u|} \sum_{i \in \mathcal{I}_u} r(u, i) \quad (6.25)$$

$$\text{std}(u) = \sqrt{\frac{1}{|\mathcal{I}_u|} \sum_{i \in \mathcal{I}_u} (r(u, i) - \text{mean}(u))^2} \quad (6.26)$$

Alternatively to these heuristic predictors, we have also experimented with a predictor defined upon the past observed recommender's performance. In this way, this predictor – denoted as **training performance** from now on – use a validation set (as a subset of the original training set) to evaluate the performance of each user with respect to a specific recommender; then, this value is the one returned by the predictor at test time. This approach is inspired in the Machine Learning techniques which aim to learn a feature (in this case, the user's performance) by using some training information. For this predictor, this training information is the performance computed on the validation set.

Additionally, we propose to measure the **entropy** of the user's preferences as a quantification of the uncertainty associated with a probability distribution (Cover and Thomas, 1991). We may therefore assess the uncertainty involved in the system's knowledge about a user's preferences by the entropy of the item distribution (the probability to choose an item) given the information in the user profile, using the ground models presented in Section 6.2.1. Hence, we define this predictor as follows:

$$\text{entropy}(u) = \sum_{i \in \mathcal{I}_u} p(i|u) \log_2 p(i|u) \quad (6.27)$$

Alternative measures from Information Theory could be used to define user-based predictors, like Information Gain (Bellogín, 2009), but we leave them out of this analysis because its application to Recommender Systems is neither clear nor principled and their predictive results are not optimal. Furthermore, other measures already proposed in the literature such as inverse user frequency (Breese et al., 1998)

and the analogous inverse item frequency (Bellogín, 2009), and other manipulations of the same concept, are also ignored here because they are simply transformations of the previously presented count predictor. Finally, the concept of power users (Lathia et al., 2008) could also be used as a proxy for well-performing users, but preliminary results have not shown strong predictive power.

### 6.4.2 Using log-based preference data

As we have observed in the previous section, recommendation performance usually has obvious predictors, obvious in the sense that they do not involve any interesting finding or insightful kind of analysis, or anything to learn from. We include in our analysis some of these obvious predictors, framed as baseline performance predictors that basically count how many interactions a user has had with the system. In this sense, these predictors are slightly different to the ones presented in the previous section, namely because in log-based datasets repetitions of items are allowed in a user's profile. In order to account for this difference, apart from **count**, **mean**, and **standard deviation** predictors, we propose to normalise the count predictor by the number of items consumed by each user, that is, we define the **average count** predictor as follows:

$$\text{average count}(u) = \frac{|\{(u, j, s) \in \mathcal{L} | j \in \mathcal{J}, s \in \mathcal{S}\}|}{|\{j \in \mathcal{L} : (u, j, s) \in \mathcal{L}, s \in \mathcal{S}\}|} \quad (6.28)$$

We also test more elaborate predictors based on the temporal dimension, such as the ones defined in (Díaz and Jones, 2004). First-order autocorrelation (or temporal self-correlation) can be considered with a reinterpretation of the random variables. Specifically, this predictor, in contrast with the temporal Kullback-Leibler divergence where the similarity with the temporal background model is assessed, captures the structure of the query time series. For instance, a uniform distribution would have an autocorrelation value of 0, whereas a query time series with strong inter-day (or whatever segment size is used to build the discrete time series) dependency will obtain a high autocorrelation value.

Thus, we define the **autocorrelation** user predictor as follows:

$$\text{autocorrelation}(u) = \frac{\sum_{t=1}^T (p(t|u) - 1/T)(p(t+1|u) - 1/T)}{\sum_{t=1}^T (p(t|u) - 1/T)^2} \quad (6.29)$$

where  $T$  is the total number of time units in the time interval. We can observe how this predictor captures the similarity between two consecutive observations.

Extensions of this predictor could use the probabilities defined in Section 6.2.2, like  $p(t|u, i)$ , instead of  $p(t|u)$ . Similarly, other predictors proposed by Díaz and Jones in (Díaz and Jones, 2004) and (Jones and Díaz, 2007) such as the kurtosis or

the burst model could be adapted to recommender systems, but we leave such extensions for future work.

## 6.5 Experimental results

In this section we provide correlation results where all the predictors – heuristic, social, and clarity-based – are compared against each other using an array of recommendation methods and evaluation methodologies.

### 6.5.1 User predictors using rating-based preference data

In this section we compare the correlations obtained for the clarity-based predictors defined in Section 6.2.1, the user entropy defined in Equation (6.27), and the baselines presented in Equations (6.24), (6.25), and (6.26) using the MovieLens 1M dataset. The  $\lambda$  parameter for the language model smoothing was not optimised for this task and a default value of 0.6 was used in all the models as originally used in (Cronen-Townsend et al., 2002). Here, we focus on Pearson’s correlation and P@10. Additional results are reported in Appendix A.4.1.

Table 6.7 shows the correlation values when the AR methodology is used. We can observe fairly high correlation values for recommenders pLSA, ItemPop, TFL2, and kNN, comparable to results in the query performance literature. A slightly lower correlation is found for TFL1, whereas no correlation is found for CB and IB. These results are consistent when other performance metrics are used such as nDCG, and at different cutoff points. Spearman’s correlation yields similar values. Here we also include the count predictor in test, which is obviously not a predictor in strict sense,

| Predictor            | Random | CB    | IB     | ItemPop | kNN    | pLSA   | TFL1  | TFL2   | Median | Mean   |
|----------------------|--------|-------|--------|---------|--------|--------|-------|--------|--------|--------|
| Count (training)     | 0.135  | 0.164 | 0.042  | 0.512   | 0.424  | 0.442  | 0.198 | 0.644  | 0.311  | 0.320  |
| Count (test)         | 0.135  | 0.172 | 0.042  | 0.520   | 0.431  | 0.452  | 0.200 | 0.647  | 0.316  | 0.325  |
| Training performance | 0.024  | 0.176 | 0.258  | 0.429   | 0.296  | 0.357  | 0.215 | 0.485  | 0.277  | 0.280  |
| Mean                 | 0.019  | 0.067 | -0.002 | 0.015   | 0.022  | 0.108  | 0.026 | -0.018 | 0.021  | 0.030  |
| Standard deviation   | 0.008  | 0.008 | 0.011  | -0.029  | -0.031 | -0.032 | 0.011 | -0.051 | -0.011 | -0.013 |
| ItemSimple Clarity   | 0.149  | 0.191 | 0.046  | 0.549   | 0.453  | 0.489  | 0.222 | 0.683  | 0.338  | 0.348  |
| ItemUser Clarity     | 0.134  | 0.166 | 0.048  | 0.493   | 0.416  | 0.428  | 0.215 | 0.634  | 0.316  | 0.317  |
| RatUser Clarity      | 0.135  | 0.160 | 0.048  | 0.514   | 0.442  | 0.435  | 0.214 | 0.651  | 0.325  | 0.325  |
| RatItem Clarity      | 0.127  | 0.159 | 0.039  | 0.475   | 0.402  | 0.405  | 0.203 | 0.611  | 0.303  | 0.303  |
| IRUser Clarity       | 0.128  | 0.157 | 0.027  | 0.486   | 0.382  | 0.408  | 0.182 | 0.599  | 0.282  | 0.296  |
| IRItem Clarity       | 0.122  | 0.165 | 0.034  | 0.446   | 0.352  | 0.386  | 0.188 | 0.551  | 0.270  | 0.281  |
| IRUserItem Clarity   | 0.128  | 0.158 | 0.033  | 0.479   | 0.379  | 0.403  | 0.193 | 0.594  | 0.286  | 0.296  |
| Entropy              | 0.121  | 0.168 | 0.025  | 0.492   | 0.389  | 0.483  | 0.140 | 0.589  | 0.279  | 0.301  |
| <b>Median</b>        | 0.128  | 0.162 | 0.037  | 0.489   | 0.396  | 0.418  | 0.196 | 0.605  |        |        |
| <b>Mean</b>          | 0.112  | 0.145 | 0.033  | 0.413   | 0.338  | 0.367  | 0.166 | 0.511  |        |        |

**Table 6.7.** Pearson’s correlation between rating-based user predictors and P@10 for different recommenders using the AR methodology (MovieLens dataset).

since it uses a different input than the other predictors, but we include it in our analysis as a further reference to check behaviours.

As mentioned in Chapter 5, the standard procedure in Information Retrieval for this kind of evaluation is to compute correlations between the predictor(s) and one retrieval model (like in (Cronen-Townsend et al., 2002) and (Hauff et al., 2008a)) or an average of several methods (Mothe and Tanguy, 2005). This approach may hide the correlation effect for some recommenders, as we may observe from the median and mean correlation values included in the table, which are still very large despite the fact that two of the recommenders analysed have much lower correlations. Nonetheless, these aggregated values, i.e., the mean and the median, provide competitive correlation values when compared with those in the literature.

The difference in correlation for CB and IB recommenders may be explained considering two factors: the actual recommender performance and the input sources used by the recommender. With regards to the first factor, as presented in Table 6.8, the IB algorithm performs poorly (in terms of the considered ranking quality metrics, such as precision and nDCG) in comparison to the rest of recommenders. It seems natural that a good predictor for a well performing algorithm (specifically, pLSA is the best performing recommender in this context) would hardly correlate at the same time with a poorly performing one.

This does not explain however the somewhat lower correlation with the content-based recommender, which has better performance than TFL1. The input information that this recommender and the predictors take in are very different: the latter compute probability distributions based on ratings given by users to items, while the former uses content features from items, such as directors and genres. Furthermore, the CB recommender is not coherent with the inherent probabilistic models described by the predictors, since the events modeled by each of them are different: CB would be related to the likelihood that an item is described by the same features as those items preferred by the user, whereas predictors are related to the probability that an item is rated by a user. Moreover, the predictors' ground models coherently fit in the standard collaborative framework (Wang et al., 2008a), which reinforces the suitability of the user performance predictors presented herein, at least for collaborative filtering recommenders.

It is worth noting to this respect that most clarity-based query performance predic-

| Recommender     | Random | CB     | IB     | ItemPop | kNN    | pLSA   | TFL1   | TFL2   |
|-----------------|--------|--------|--------|---------|--------|--------|--------|--------|
| AR methodology  | 0.0025 | 0.0163 | 0.0001 | 0.0897  | 0.0307 | 0.1454 | 0.0024 | 0.0696 |
| 1R methodology  | 0.0099 | 0.0221 | 0.0074 | 0.0649  | 0.0437 | 0.0836 | 0.0221 | 0.0690 |
| U1R methodology | 0.0100 | 0.0223 | 0.0068 | 0.0406  | 0.0381 | 0.0718 | 0.0294 | 0.0524 |
| P1R methodology | 0.0101 | 0.0197 | 0.0208 | 0.0282  | 0.0265 | 0.0604 | 0.0203 | 0.0348 |

**Table 6.8. Summary of recommender performance using different evaluation methodologies (evaluation metric is P@10 with the MovieLens dataset).**

tion methods in Information Retrieval study their predictive power on language modelling retrieval systems (Cronen-Townsend et al., 2002; Hauff et al., 2008a; Zhou and Croft, 2007) or similar approaches (He and Ounis, 2004). This suggests that a well performing predictor should be defined upon common spaces, models, and estimation techniques as the retrieval system the performance of which is meant to be predicted.

Finally, the correlation values found by the training performance predictors, although sometimes strong, are not as high as those of the baselines predictors – such as training count – in most situations, in particular, they are always lower except for the IB and TFL1 recommenders. This highlights the importance of having a more general model for predicting the performance of a user, since these predictors in fact depend considerably on the properties of the validation (and test) partition of the data, such as the amount of sparsity, type of items evaluated and so on.

### Unbiased performance prediction

In Chapter 4 we already demonstrated that some methodologies may be biased towards more popular items or sparsity constraints. We can observe in the previous table that trivial predictors such as count (either in training or in test) obtain significant (and positive) correlation, no matter the recommender. We argue whether this is because these predictors are really capturing an interesting effect or the evaluation methodology is prone to such effect. In order to overcome this problem, now we present the same correlation analysis but with the different methodologies presented in Chapter 4.

In Table 6.9 we show results with the methodology 1R. Here we can observe that most of the correlation values are lower than in the previous case; interestingly, the correlation with the Random recommender now is almost 0 for every predictor (and in particular, for the training and test profile size). This is evidence that per-

| Predictor            | Random | CB     | IB     | ItemPop | kNN    | pLSA   | TFL1   | TFL2   |
|----------------------|--------|--------|--------|---------|--------|--------|--------|--------|
| Count (training)     | 0.061  | -0.038 | 0.092  | 0.258   | 0.108  | 0.303  | 0.086  | 0.394  |
| Count (test)         | 0.063  | -0.033 | 0.091  | 0.266   | 0.115  | 0.312  | 0.089  | 0.398  |
| Training performance | 0.012  | 0.332  | 0.168  | 0.272   | 0.266  | 0.133  | 0.303  | 0.240  |
| Mean                 | 0.036  | 0.082  | -0.029 | 0.028   | 0.111  | 0.117  | 0.145  | 0.031  |
| Standard deviation   | -0.010 | 0.006  | 0.051  | -0.060  | -0.116 | -0.080 | -0.040 | -0.114 |
| ItemSimple Clarity   | 0.066  | -0.033 | 0.094  | 0.265   | 0.115  | 0.322  | 0.105  | 0.409  |
| ItemUser Clarity     | 0.059  | -0.038 | 0.087  | 0.236   | 0.100  | 0.287  | 0.096  | 0.375  |
| RatUser Clarity      | 0.057  | -0.054 | 0.083  | 0.245   | 0.130  | 0.285  | 0.086  | 0.372  |
| RatItem Clarity      | 0.057  | -0.044 | 0.069  | 0.225   | 0.110  | 0.268  | 0.094  | 0.352  |
| IRUser Clarity       | 0.056  | -0.020 | 0.053  | 0.250   | 0.069  | 0.280  | 0.077  | 0.364  |
| IRItem Clarity       | 0.051  | -0.010 | 0.058  | 0.205   | 0.029  | 0.235  | 0.074  | 0.310  |
| IRUserItem Clarity   | 0.056  | -0.020 | 0.052  | 0.242   | 0.066  | 0.273  | 0.081  | 0.357  |
| Entropy              | 0.091  | 0.021  | 0.144  | 0.354   | 0.169  | 0.460  | 0.114  | 0.543  |

**Table 6.9. Pearson's correlation between rating-based user predictors and P@10 for different recommenders using the 1R methodology (MovieLens dataset).**

| Predictor          | Random | CB     | IB     | ItemPop | kNN    | pLSA   | TFL1   | TFL2   |
|--------------------|--------|--------|--------|---------|--------|--------|--------|--------|
| Count (training)   | 0.048  | -0.012 | 0.237  | 0.162   | 0.115  | 0.140  | 0.022  | 0.235  |
| Count (test)       | 0.049  | -0.001 | 0.226  | 0.135   | 0.110  | 0.137  | 0.036  | 0.213  |
| Mean               | 0.023  | 0.051  | -0.035 | 0.009   | 0.108  | 0.075  | 0.155  | -0.006 |
| Standard deviation | 0.015  | 0.032  | 0.023  | -0.047  | -0.098 | -0.038 | -0.061 | -0.049 |
| ItemSimple Clarity | 0.055  | -0.005 | 0.241  | 0.166   | 0.128  | 0.153  | 0.042  | 0.241  |
| ItemUser Clarity   | 0.046  | -0.009 | 0.232  | 0.142   | 0.109  | 0.133  | 0.028  | 0.216  |
| RatUser Clarity    | 0.045  | -0.028 | 0.234  | 0.155   | 0.137  | 0.130  | 0.022  | 0.225  |
| RatItem Clarity    | 0.043  | -0.025 | 0.212  | 0.136   | 0.119  | 0.117  | 0.033  | 0.203  |
| IRUser Clarity     | 0.044  | 0.002  | 0.180  | 0.153   | 0.069  | 0.134  | 0.029  | 0.210  |
| IRItem Clarity     | 0.036  | 0.011  | 0.178  | 0.114   | 0.035  | 0.108  | 0.014  | 0.173  |
| IRUserItem Clarity | 0.042  | 0.003  | 0.178  | 0.147   | 0.065  | 0.130  | 0.028  | 0.203  |
| Entropy            | 0.078  | 0.044  | 0.278  | 0.227   | 0.169  | 0.249  | 0.073  | 0.321  |

**Table 6.10.** Pearson's correlation between rating-based user predictors and P@10 for different recommenders using the U1R methodology (MovieLens dataset).

formance results using the AR methodology are higher for users with more items in their test, independently from the recommendation algorithm complexity (see correlations with Random recommender in Table 6.7). In the same way, the U1R (Table 6.10) and P1R (Table 6.11) methodologies also obtain negligible correlation values for the Random recommender, which confirms the suitability of these methodologies for our purposes. We also have to note that we have not applied the training performance predictor in these methodologies because their restrictions do not let to replicate the same conditions in a validation split. Furthermore, as stated in Chapter 4, both approaches aim to remove the bias towards more popular items. Here, we can observe how the correlation with respect to the ItemPop recommender is comparable to that with the Random recommender with the P1R methodology, confirming again the ability of this methodology to produce unbiased results (at least, with respect to popular items).

The main difference in the results obtained between these three methodologies (1R, U1R, and P1R) seems to be more at the recommender level rather than at the

| Predictor          | Random | CB     | IB     | ItemPop | kNN    | pLSA   | TFL1   | TFL2   |
|--------------------|--------|--------|--------|---------|--------|--------|--------|--------|
| Count (training)   | 0.073  | -0.005 | 0.253  | 0.088   | 0.103  | 0.160  | -0.001 | 0.307  |
| Count (test)       | 0.076  | 0.000  | 0.253  | 0.093   | 0.108  | 0.168  | 0.003  | 0.308  |
| Mean               | 0.034  | 0.073  | -0.033 | 0.008   | 0.110  | 0.085  | 0.188  | -0.026 |
| Standard deviation | -0.010 | 0.009  | 0.014  | -0.058  | -0.104 | -0.044 | -0.061 | -0.051 |
| ItemSimple Clarity | 0.078  | 0.000  | 0.254  | 0.084   | 0.111  | 0.169  | 0.019  | 0.313  |
| ItemUser Clarity   | 0.072  | -0.001 | 0.249  | 0.075   | 0.101  | 0.156  | 0.005  | 0.303  |
| RatUser Clarity    | 0.071  | -0.016 | 0.252  | 0.086   | 0.128  | 0.148  | 0.003  | 0.297  |
| RatItem Clarity    | 0.067  | -0.011 | 0.234  | 0.077   | 0.113  | 0.138  | 0.016  | 0.288  |
| IRUser Clarity     | 0.066  | 0.002  | 0.200  | 0.086   | 0.066  | 0.147  | 0.006  | 0.274  |
| IRItem Clarity     | 0.059  | 0.010  | 0.192  | 0.061   | 0.037  | 0.123  | -0.006 | 0.242  |
| IRUserItem Clarity | 0.066  | 0.003  | 0.200  | 0.082   | 0.065  | 0.145  | 0.006  | 0.272  |
| Entropy            | 0.092  | 0.038  | 0.286  | 0.133   | 0.128  | 0.266  | 0.039  | 0.379  |

**Table 6.11.** Pearson's correlation between rating-based user predictors and P@10 for different recommenders using the P1R methodology (MovieLens dataset).

predictor level, in the sense that the trend in predictor effectiveness is similar for each methodology but the correlations obtained for each recommender vary dramatically from one methodology to another. For instance, IB recommender obtains near zero correlations with 1R but higher (significative) values for U1R and P1R; a similar situation occurs with the TFL2 recommender, where the correlations are lower for the U1R methodology and higher for 1R and P1R. Note that the training and test sets are the same for all the methodologies except for U1R, which means that the performance predictors are entirely new for that methodology. Thus, *a priori* it would not be clear that such an agreement between the different methodologies should appear at the predictor level unless they are really capturing the same nuance about the user, no matter the evaluation methodology used.

It is worth noting that the correlation values of these three methodologies have been found after a careful examination of the available data, where two different trends emerged: one where the performance values were more or less uniformly distributed in the interval  $[0, 0.1]$  – recall that 0.1 is the maximum value for the metric P@10 with the 1R methodology, since there is only one relevant item – ; and a second one where a fixed value was obtained. This second trend, against which our predictors shown no correlation at all (since the performance had a zero standard deviation, and thus the correlation was impossible to calculate) is able to degrade the correlation coefficient almost to negligible values, mainly because it accounts for half of the number of points. This problem with correlation coefficients, and with Pearson's correlation in particular, is well known in the literature of performance prediction (Hauff, 2010; Pérez Iglesias, 2012). For this reason, we have divided the performance values and computed two correlations in order to account for these two trends: the values with respect to the first trend are those presented in the previous tables, whereas the correlation with respect to the second trend was not computable because the variable had a zero standard deviation.

In summary, there seems to be no clear winner among the set of performance predictors proposed. The predictive power of each of them is clearly influenced by the actual recommender its performance aims to be predicted and the evaluation methodology in use. Nonetheless, **the proposed predictors usually obtain higher correlation values than baseline predictors** such as the mean or the standard deviation, evidencing their predictive power **independently from the evaluation methodology**. Surprisingly, the ItemSimple clarity predictor obtains very good results in most of the situations, although more complex predictors like IRUser or IRUserItem clarity obtain stronger correlations for some recommenders.

### 6.5.2 Item predictors using rating-based preference data

In the same way we have assessed the predictive power of user predictors, we now aim to estimate the predictive power of item predictors. However, the true perform-

| $u_1$       | $u_2$       | $u_3$       | $i_1$       | $i_2$     | $i_3$       | $i_4$       |
|-------------|-------------|-------------|-------------|-----------|-------------|-------------|
| $i_1$ 0.8   | $i_2$ 0.6   | $i_3$ 0.9   | * $u_1$ 0.8 | $u_1$ 0.7 | $u_3$ 0.9   | * $u_3$ 0.6 |
| * $i_2$ 0.7 | * $i_3$ 0.5 | * $i_4$ 0.6 | * $u_3$ 0.5 | $u_2$ 0.6 | * $u_1$ 0.6 | $u_1$ 0.5   |
| * $i_3$ 0.6 | * $i_4$ 0.4 | * $i_1$ 0.5 | $u_2$ 0.3   | $u_3$ 0.1 | * $u_2$ 0.5 | * $u_2$ 0.4 |
| $i_4$ 0.5   | $i_1$ 0.3   | $i_2$ 0.1   |             |           |             |             |
| P@2         | 0.5         | 0.5         | 1.0         | 0.0       | 0.5         | 0.5         |

**Table 6.12. Procedure to obtain ranking for items from user rankings generated by a standard recommender. \* denotes a relevant item, and the numbers are the score predicted by the recommendation method.**

ance value for an item is not straightforward to compute, since the process has to produce unbiased results in the space of items (as described in Chapter 4) but with the characteristic that the item dimension is not the main input of the recommendation process, and thus, some new approach has to be put in place.

There are basically two possibilities for computing the true performance on an item: either starting from the results obtained using a standard procedure (obtain a ranking for each user by recommending items to users), then transposing users and items (generating, thus, user rankings for each item) and computing the per-ranking performance as usual; or transpose the original rating matrix in order to effectively “recommend users” for each item. This would implicitly imply a transposition of the recommendation task, which may also make sense: find the most suitable users to recommend an item – this would be the scenario, e.g. in advertisement targeting when a new product is released on the market. Here, we use the former approach since the latter does not produce consistent results in our experiments, probably because the recommendation problem is not completely symmetric and, thus, this method is not able to properly capture the recommender’s performance for each item. On the other hand, non-personalised recommenders (such as recommendation by item popularity) cannot be applied in the symmetric formulation: since the same item ranking is built for all users, the user ranking for an item would be a global tie on all users. Table 6.12 shows an example of how we may transpose users and items from an item ranking for three users. We show that the precision for all the users is the same, whereas for the items is completely diverse, ranging from zero to perfect precision.

In our experiments, we have tested the different methodologies already presented along with a modified version of the U1R evaluation methodology (user-uniform U1R, or uuU1R). The rationale for the uuU1R design goes as follows: in the U1R methodology we force the same number of ratings (or, equivalently, users) for the items in the test set, however, users are freely assigned to each item. Now, when we transpose users and items this situation may produce a new problem, since there

| Predictor           | Random | CB     | ItemPop | kNN    | pLSA   |
|---------------------|--------|--------|---------|--------|--------|
| Count (training)    | 0.414  | 0.060  | -0.151  | -0.021 | -0.269 |
| Count (test)        | -----  | -----  | -----   | -----  | -----  |
| Mean                | 0.602  | 0.125  | 0.096   | 0.040  | -0.038 |
| Standard deviation  | -0.313 | 0.025  | -0.006  | -0.003 | 0.075  |
| UserSimple Clarity  | 0.467  | 0.080  | -0.120  | -0.015 | -0.240 |
| UserItem Clarity    | 0.419  | 0.064  | -0.145  | -0.018 | -0.261 |
| RatItem Clarity     | 0.440  | 0.075  | -0.127  | -0.015 | -0.230 |
| RatUser Clarity     | 0.451  | 0.085  | -0.103  | -0.004 | -0.201 |
| URItem Clarity      | 0.396  | 0.053  | -0.174  | -0.026 | -0.289 |
| URUser Clarity      | 0.408  | 0.072  | -0.132  | -0.004 | -0.243 |
| URIItemUser Clarity | 0.409  | 0.061  | -0.161  | -0.021 | -0.277 |
| Entropy             | 0.381  | -0.001 | -0.216  | -0.055 | -0.442 |

**Table 6.13. Pearson’s correlation for rating-based item predictors and precision using the uuU1R methodology (MovieLens dataset).**

could be users assigned to more items which would bias the ranking’s performance towards items contained in the test set of heavy raters. Therefore, if we impose a uniform distribution also on the user’s dimension, this bias should decrease. We refer to the reader to Appendix A.3 for more details.

However, despite these efforts, we have not found a reliable methodology to evaluate the item performance. We present in Table 6.13 the results using the uuU1R methodology and the predictors defined in Table 6.5 for the precision metric. Recall that, since we transpose users and items from the generated rankings, to obtain a similar measure of P@10 we only use the top 10 items from each original ranking and then compute precision over the whole ranking for each item. We may observe in the table that the correlations with the Random recommender are very strong, questioning the validity of such results. Besides, the entropy predictor obtains stronger correlation than clarity-based in this case, and most of them (except for URItem) show little difference to training count. Note that it is not possible to compute a correlation with the test count predictor since that predictor has a constant value with zero standard deviation (see Equation (5.11) for more details on Pearson’s correlation) since every item has the same number of ratings in the test set in the uuU1R methodology.

As a conclusion, we have found that **a proper evaluation of item performance is not obvious**, mainly because the task of suggesting users to items is not completely symmetric with respect to the standard task of recommendation. We have devised different methodologies to estimate the recommendatioin performance of an item, however the difficulty lies mainly in forming consistent lists of “recommended” users for items, a difficulty which is not conceptual (ranking target users to whom an item may be recommended does make sense as a task in many scenarios), but technical (obtaining balanced result lists that allow for undistorted performance measurements).

### 6.5.3 User predictors using log-based preference data

In this section we analyse the correlation obtained between the predictors defined in Sections 6.2.2 and 6.4.2 and five recommenders using the 1R methodology on two versions of the Last.fm dataset – one where a temporal partition is performed and another where the partition is randomly made (more details about the splits in Appendix A.1.2). No smoothing was used in the language models since preliminary tests obtained better results with lower values of  $\lambda$ . Besides, for comparison purposes, we also include one of the clarity models proposed for rating-based preference data using the transformation proposed in Section 6.2.2 to use such predictors with log data along with the frequency-based clarity proposed in Equation (6.20). Like in the previous section, Pearson’s correlation with the P@10 evaluation metric is reported; for additional metrics, see Appendix A.4.2.

First, we can observe in Table 6.14 (temporal split) that ItemPriorTime clarity obtains strong correlation values, especially for the ItemPop and kNN recommenders. It is interesting to compare the correlations between this predictor and the Item-Time clarity, which are much lower. This is probably because the ItemPriorTime clarity predictor, as opposed to ItemTime clarity, incorporates a component that measures the item popularity, i.e.,  $p(i)$ . The TimeSimple and the frequency-based clarity predictors, on the other hand, obtain strong correlation but negative values for all the recommenders except the ItemPop for the TimeSimple predictor. Furthermore, the ItemSimple clarity (a predictor based on explicit information) obtains negligible correlations except for the ItemPop and kNN recommenders.

Table 6.15, on the other hand, shows the results when a random split is used. We have to note that such split does not preserve the temporal continuity of the user’s preferences, and thus, any recommender or technique which makes use of temporal features is not guaranteed to succeed. Here, we can observe that TimeSimple predictor obtains strong correlations for all the recommenders except for the Random

| Predictor               | Random | CB     | ItemPop | kNN    | pLSA   |
|-------------------------|--------|--------|---------|--------|--------|
| Average Count           | 0.027  | 0.138  | 0.069   | -0.013 | 0.191  |
| Count                   | 0.046  | 0.118  | -0.058  | 0.131  | 0.139  |
| Mean                    | -0.079 | -0.361 | 0.054   | -0.110 | -0.376 |
| Standard deviation      | -0.050 | -0.158 | 0.082   | -0.132 | -0.177 |
| Autocorrelation         | 0.004  | 0.139  | -0.066  | -0.105 | 0.100  |
| TimeSimple Clarity      | -0.091 | -0.342 | 0.093   | -0.317 | -0.354 |
| ItemTime Clarity        | 0.037  | 0.078  | 0.038   | 0.258  | 0.064  |
| ItemPriorTime Clarity   | 0.057  | 0.154  | 0.189   | 0.307  | 0.154  |
| Frequency-based Clarity | -0.049 | -0.410 | -0.221  | -0.291 | -0.376 |
| ItemSimple Clarity      | 0.027  | 0.047  | -0.107  | 0.221  | 0.029  |

**Table 6.14. Pearson’s correlation between log-based predictors and P@10 for different recommenders using 1R methodology (Last.fm temporal dataset).**

| Predictor               | Random | CB     | ItemPop | kNN    | pLSA   |
|-------------------------|--------|--------|---------|--------|--------|
| Average Count           | -0.023 | -0.068 | -0.170  | -0.018 | -0.087 |
| Count                   | -0.012 | -0.236 | -0.242  | -0.086 | -0.198 |
| Mean                    | 0.036  | 0.182  | 0.100   | 0.047  | 0.118  |
| Standard deviation      | -0.009 | 0.089  | 0.079   | 0.092  | 0.082  |
| Autocorrelation         | 0.045  | -0.069 | -0.089  | -0.012 | -0.055 |
| TimeSimple Clarity      | 0.031  | 0.274  | 0.314   | 0.169  | 0.240  |
| ItemTime Clarity        | 0.021  | -0.145 | 0.004   | 0.025  | -0.053 |
| ItemPriorTime Clarity   | 0.011  | -0.057 | 0.176   | 0.145  | 0.083  |
| Frequency-based Clarity | 0.025  | 0.018  | -0.287  | -0.182 | -0.220 |
| ItemSimple Clarity      | 0.020  | -0.247 | -0.163  | -0.068 | -0.186 |

**Table 6.15. Pearson's correlation between log-based predictors and P@10 for different recommenders using 1R methodology (Last.fm five-fold dataset).**

technique. Like before, ItemPriorTime has a high correlation with the ItemPop recommender. In contrast with the previous situation, the ItemSimple clarity obtains strong but negative correlations for the personalised recommenders. Besides, the frequency-based clarity has negative correlations for all the recommenders except CB, a consistent situation with the results obtained with the temporal split.

Hence, we may conclude that **log-based and time-aware predictors successfully predict the performance of the recommendation algorithms**, although in some situations the sign of the prediction is negative. Moreover, frequency-based, ItemSimple, and TimeSimple clarity obtain consistently strong correlations both in a temporal split and in a random split of the data, evidencing their predictive power.

#### 6.5.4 User predictors using social-based preference data

In this section we study the correlation between the predictors described in Section 6.3 and several recommenders using the two versions of the CAMRa dataset: social and collaborative. In this case, we also consider social filtering recommenders in order to analyse whether these predictors are sensitive to the source of information used by the recommender, and thus, whether they obtain stronger correlations with social filtering recommenders. Besides, one clarity-based predictor (ItemSimple) and the baseline rating predictors presented in Section 6.4.1 are also included in the analysis for comparison purposes. Additionally, for the HITS and PageRank graph metrics in this experiment we use the implementation developed in the JUNG library (O'Madadhain et al., 2003).

Table 6.16 shows correlation values obtained when using the AR methodology in the social version of the dataset. Here, we can observe that most of the correlation values obtained for the social predictors are negative, representing that the lower the predictor output, the better the performance, which may seem a little counter-intuitive, at least for the social filtering recommenders (Personal and PureSocial).

Among the social-based predictors, degree and two-hop neighbourhood size obtain better correlations than the rest.

A similar situation is presented in Table 6.17, where the collaborative-social version of the dataset is used. Again, most of the correlations with the social-based predictors are negative, and degree and two-hop neighbourhood size obtain higher correlations (in absolute value). Interestingly, in this situation strong correlations are obtained with the user-based recommender (kNN), in particular with degree and the average neighbour degree predictors. Nonetheless, these correlations are lower than those obtained for the ItemSimple predictor with the collaborative filtering recommenders. At the same time, this predictor always obtains worse correlations (in absolute value) than the social-based predictors for the social filtering recommenders, as expected.

Additionally, note that the number of points used in the correlation computation is different in each version of the dataset, namely: in the collaborative-social version

| Predictor              | Random | ItemPop | kNN    | pLSA   | Personal | PureSocial |
|------------------------|--------|---------|--------|--------|----------|------------|
| Count (training)       | 0.032  | 0.122   | 0.113  | 0.031  | 0.062    | 0.111      |
| Count (test)           | 0.158  | 0.252   | 0.382  | 0.167  | 0.235    | 0.174      |
| Mean                   | -0.066 | 0.033   | -0.012 | 0.023  | -0.057   | -0.051     |
| Standard deviation     | 0.034  | 0.054   | -0.020 | 0.115  | 0.128    | 0.183      |
| Avg neighbour degree   | -0.062 | -0.089  | -0.013 | 0.011  | -0.074   | -0.106     |
| Betweenness centrality | -0.031 | -0.016  | 0.027  | -0.038 | -0.012   | -0.079     |
| Clustering coefficient | 0.049  | -0.084  | -0.023 | 0.048  | -0.027   | -0.035     |
| Degree                 | -0.038 | -0.046  | 0.015  | -0.059 | -0.147   | -0.133     |
| Ego components size    | -0.058 | 0.005   | 0.004  | -0.046 | -0.056   | -0.020     |
| HITS                   | -0.021 | -0.043  | 0.005  | 0.061  | 0.038    | 0.000      |
| PageRank               | -0.022 | -0.025  | -0.023 | -0.039 | -0.102   | -0.037     |
| Two-hop neighbourhood  | -0.080 | -0.082  | 0.004  | -0.054 | -0.123   | -0.136     |
| ItemSimple Clarity     | 0.030  | 0.157   | 0.130  | 0.050  | 0.072    | 0.126      |

**Table 6.16. Pearson's correlation between social-based predictors and P@10 for different recommenders using AR methodology (CAMRa Social).**

| Predictor              | Random | ItemPop | kNN    | pLSA   | Personal | PureSocial |
|------------------------|--------|---------|--------|--------|----------|------------|
| Count (training)       | 0.012  | 0.098   | 0.203  | 0.107  | 0.058    | 0.111      |
| Count (test)           | 0.096  | 0.207   | 0.389  | 0.179  | 0.232    | 0.170      |
| Mean                   | -0.067 | 0.000   | -0.126 | -0.024 | -0.051   | -0.050     |
| Standard deviation     | 0.082  | 0.014   | -0.029 | 0.016  | 0.129    | 0.182      |
| Avg neighbour degree   | 0.071  | -0.008  | 0.152  | 0.046  | -0.073   | -0.104     |
| Betweenness centrality | -0.007 | -0.008  | 0.010  | -0.005 | -0.012   | -0.078     |
| Clustering coefficient | 0.006  | -0.022  | 0.152  | 0.076  | -0.032   | -0.035     |
| Degree                 | 0.032  | 0.018   | 0.164  | 0.006  | -0.143   | -0.134     |
| Ego components size    | 0.026  | 0.044   | 0.133  | 0.002  | -0.053   | -0.022     |
| HITS                   | -0.011 | -0.034  | -0.001 | 0.061  | 0.038    | 0.001      |
| PageRank               | -0.002 | 0.021   | 0.118  | 0.014  | -0.099   | -0.040     |
| Two-hop neighbourhood  | 0.059  | -0.015  | 0.130  | 0.012  | -0.121   | -0.135     |
| ItemSimple Clarity     | 0.010  | 0.120   | 0.211  | 0.129  | 0.070    | 0.126      |

**Table 6.17. Pearson's correlation between social-based predictors and P@10 for different recommenders using AR methodology (CAMRa Collaborative).**

the number of users contained in the test set is twice the number available in the social version (see Appendix A.1.3), which means that significant correlations can be achieved with lower values (as described in Chapter 5).

In the results described above, we can observe how, like in the previous sections, the size of the user profile in test (predictor count in test) obtains significant correlations. This trend, however, is almost neutralised in the collaborative-social dataset with respect to the Random recommender. Thus, as before, we would attempt to use the 1R methodology with each dataset in order to obtain unbiased correlations towards users with more ratings in test. However, due to the lack of coverage of Personal and PureSocial recommenders, this methodology do not obtain sensible results (for instance, the value of precision at 10 is almost invariably 0.10, that is, the maximum possible value when only one relevant document – as assumed in the 1R methodology – is retrieved in the top 10, mainly because the recommender is not able to retrieve most of the not relevant items). This lack of coverage is natural for these recommenders since they can only suggest items rated by users in the active user's social network (see Appendix A.2 for details on the implementation of the algorithms).

In conclusion, most of the social performance predictors proposed obtain significant correlations, however, **correlations with the social filtering methods are not so strong as we would expect**. Nonetheless, the **ItemSimple clarity does obtain significant correlations** with respect to most of the recommenders, highlighting the importance and validity of this predictor even when the main input of some recommenders (social network) is so different to that of the predictor (ratings).

### 6.5.5 Discussion

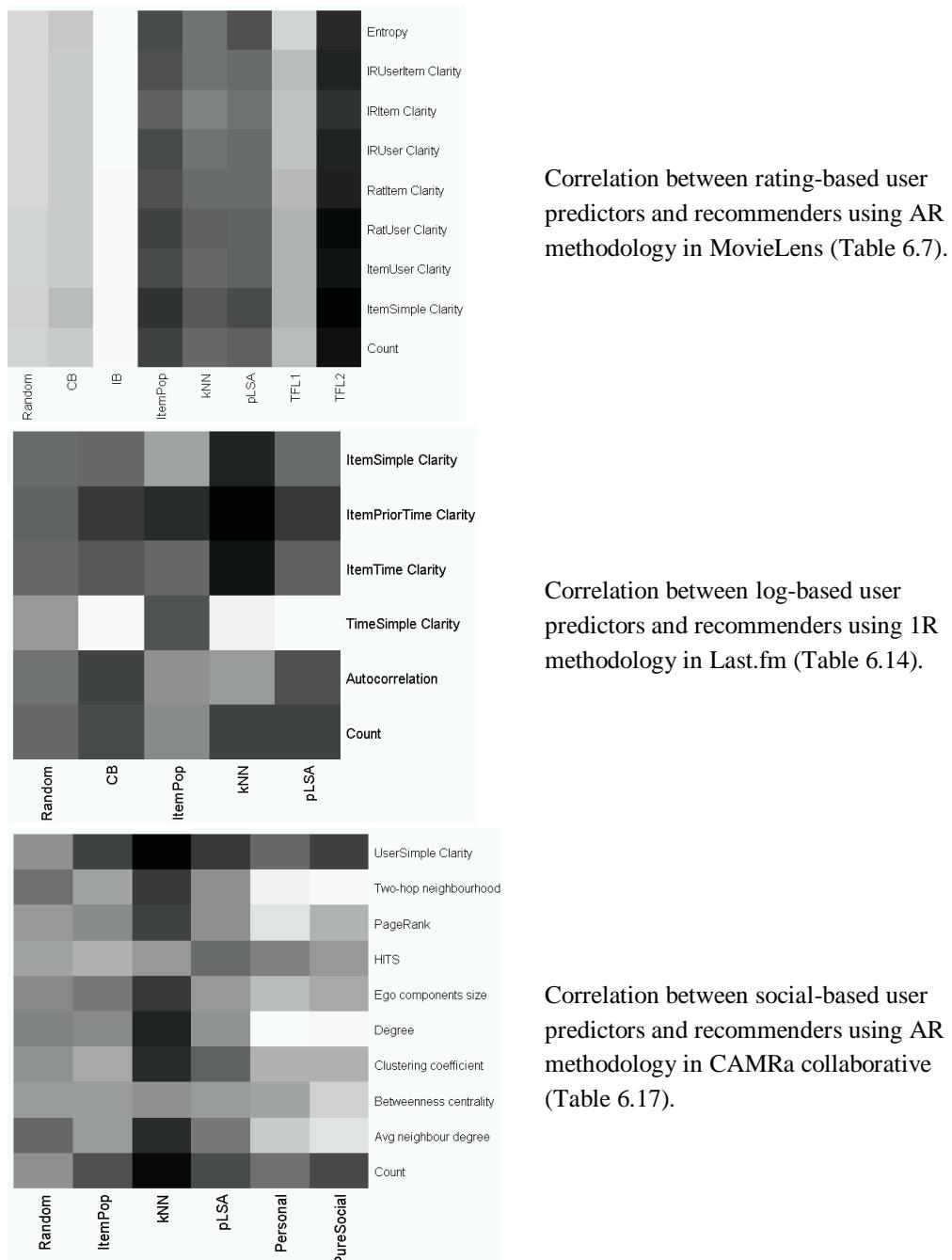
The reported experiments confirm that it is possible to predict a recommender's performance and obtain strong correlations in this regard. The results show that, in general, the proposed predictors (mostly based on Kullback-Leibler divergences over different language models and other concepts from Social Graphs and Information Theory such as entropy) obtain significant correlations in the three spaces considered: ratings, logs, and social networks. More importantly, these correlations are stronger than those obtained by more simple predictors, such as the profile size of a user, the standard deviation of her ratings, and the user's performance using a validation split. Specifically, for each recommendation input considered we have observed the following:

- Clarity-based predictors are very powerful for rating-based preferences, in particular, the ItemSimple, IRUser, and IRUserItem clarity predictors obtain strong correlations for most of the recommendation methods.

- The use of the item space as a contextual variable shows strong correlation values when the AR methodology is used, but these correlations decrease when we use unbiased methodologies, which may indicate that this new dimension is in fact capturing the item popularity and, thus, when the popularity bias is neutralised such predictors show less predictive power. We find a similar situation with the item clarity and the user space used as the contextual dimension.
- Temporal and log-based versions of the clarity predictor show higher prediction power than the rest of predictors.
- Social-based predictors are not the ones with the strongest correlation regarding the social filtering recommenders in this experiment, but the correlation found is significative and they could serve as a complement to other predictors based on a different input such as the rating-based.
- The ItemSimple clarity predictor consistently obtains strong correlation values in most of the datasets where we have analysed it. This is an evidence of the theoretical power of the user clarity to capture the uncertainty in user's tastes, even when the recommender's input is different (social filtering recommenders) or when we apply some transformation to the data (frequency-based clarity with transformation from implicit to explicit).
- As described in the Appendix A.4, most of the correlations presented in this chapter are stable when other evaluation metrics and correlation coefficients are used.

In the Recommender Systems field there are, however, additional problems due to subtle differences with respect to the common settings and experimental assumptions in Information Retrieval. Since we aim to predict the performance of a recommender, we have to be sure that we are using an unbiased performance metric, and its subsequent evaluation methodology. As we analysed in Chapter 4 there are at least two biases in the evaluation of recommender systems which may distort the results: data sparsity and item popularity. Thus, in this chapter we have computed correlations between the output of the predictors and the evaluation metrics using different evaluation methodologies, in order to analyse how sensitive the different proposed predictors are to these biases. Interestingly, although the correlations may change drastically when different evaluation methodologies are considered, most of the performance predictors still obtain good correlations. In particular, this result evidences that our proposed predictor are not so prone to the analysed biases like other simple predictors.

Finally, in Figure 6.3 we summarise the correlations found for the proposed predictors in each dimension – ratings, logs, and social. We have selected the most representative evaluation methodology (AR for rating and social data, and 1R for log



**Figure 6.3. Heatmap of the correlation values between a subset of predictors and recommenders, using the most representative methodologies for the three considered spaces.**

data) and a subset of the evaluated predictors and recommenders from each experiment, where the same information presented in Table 6.7, Table 6.14, and Table 6.17 (except for the average and median correlation values) is depicted in a more visual form. In particular, we may observe that predictors in MovieLens seem to be more redundant since the correlations are too similar.

From the figure we may also observe that in Last.fm and CAMRa datasets such redundancy is much lower and the predictors are quite different. Moreover, the first column and row (from the bottom) represent the recommender and predictor baselines, which serve as references from where the correlations should be analysed. In the three cases we can observe that most of the predictors obtain larger (darker) values than the count predictor. In the first case (rating-based predictors), however, it is clear that the correlation depends more on the recommender and less on the actual predictor.

## 6.6 Conclusions

We have proposed adaptations of query performance techniques from ad-hoc Information Retrieval to define performance predictors in Recommender Systems. Taking inspiration in the query predictor known as query clarity, we have defined and elaborated in the Recommender Systems domain several predictive models according to different formulations and assumptions. Furthermore, we propose performance predictors from theories and models of Information Theory, Social Graph Theory, and Information Retrieval based on three types of preference data: rating-based, log-based, and social-based.

We find several effective schemes with a high predictive power for recommender systems performance. We have proposed different ways for the adaptation of the query clarity predictor to recommender systems depending on the equivalences between the involved spaces. The clarity formulation is powerful because of its theoretical soundness, which is suitable to different domain-oriented adaptations. Hence, for rating-based preferences we use different expansions which take into account the rating values and the items rated by the user. For log-based preferences we exploit the co-occurrences of the items in the user profile and, more importantly, the temporal dimension, which allows for more principled functions such as the temporal Kullback-Leibler divergence or the user's autocorrelation. Finally, for social-based preferences we exploit the user's social network and different graph metrics are used apart from the user clarity based on the ratings. The results, as summarised in the previous section, are in general positive and provide evidences that the proposed functions are able to indeed predict the performance of user or items in recommender systems.

Furthermore, by analysing the behaviour of trivial predictors (such as the count of ratings in training and test) we have been able to uncover noisy biases or sensitivity to irrelevant variables in the way performance is measured. Irrelevant and uninteresting in the sense that it is not clear that the variations due to these variables really reflect actual differences in quality. As a result, we have used unbiased evaluation

methodologies where non trivial predictors still obtain positive results with respect to performance correlation.

As a side-effect, our study introduces an interesting revision of the gray sheep user concept. A simplistic interpretation of the gray sheep intuition would suggest that users with a too unusual behavior are a difficult target for recommendations. It appears however in our study that, on the contrary, users who somewhat distinguish themselves from the main trends in the community are easier to give well-performing recommendations. This suggests that perhaps the right characterisation of a gray sheep user might be one who has scarce overlap with other users. On the other hand, the fact that a clear user distinguishes herself from the aggregate trends does not mean that she does not have a sufficiently strong neighbourhood of similar users. In particular, this seems to indicate that users who follow mainstream trends are more difficult to be suggested successful items by a recommender system (at least, by a personalised one). In Information Retrieval, one can observe a similar trend: more ambiguous (mixture of topics) queries perform worse than higher-coherence queries (Cronen-Townsend et al., 2002).

In the future we plan to explore further performance predictors. Specifically, we are interested in incorporating explicit recommender dependence into the predictors, so as to better exploit the information managed by the recommender, allowing to the predictor a smoother adaptation to the recommender performance, and increasing the final correlation between them. Additionally, we are also interested in exploring alternative item-based predictors apart from those defined in this chapter, and, eventually, using other information sources such as log-based preference data and even the social network of the users who rated a particular item.



## Part IV

# Applications

*It is through science that we prove, but  
through intuition that we discover.*

Jules Henri Poincaré



# Chapter 7

## Dynamic recommender ensembles

Hybrid recommender systems – and recommender ensembles as a particular case – have become a very popular strategy for making recommendations, since they help alleviate most of the shortcomings of the individual recommenders combined. They have, however, specific problems such as the need of deciding which information sources should be exploited, which recommenders should exploit each of these sources, and how the combination of recommenders should be configured.

In this chapter we propose a framework to decide how dynamic hybridisation should be balanced, by estimating its expected improvements on individual recommendations. Furthermore, we provide some requirements to decide when to build such hybridisation. Within the spectrum of hybrid recommendation approaches, we focus on those that linearly combine the output from several recommenders, and use different weights for generating a particular aggregation of the individual recommendations. In the standard approach, these weights are typically fixed regardless of the user for which recommendations are produced, or the recommended items. In this context we investigate the use of performance predictors to assign those weights dynamically depending on the target user or item. We evaluate our approach using the predictors proposed in the previous chapter. The results obtained show that the generated dynamic ensembles are capable of outperforming their static counterparts. Furthermore, they also show that dynamic ensembles can be improved if predictors with stronger predictive power (higher correlation values as observed in the previous chapter) are used.

In Section 7.1 we present and formulate the research problem of recommendation hybridisation. Next, in Section 7.2 we describe our proposed performance prediction framework for dynamic hybrid recommendation. Section 7.3 describes the experiments conducted and provide an overall discussion of the obtained results. Finally, in Section 7.4 some conclusions are given.

## 7.1 Problem statement

As described in Chapter 2, hybrid recommenders are built by the combination of different recommendation methods. In the simplest and typical case, hybrid recommendations are produced by weighting and summing the utility values output by some recommenders, forming a so called recommender ensemble where an arbitrary number of algorithms of different kinds (content-based, user-based collaborative filtering, item-based collaborative filtering, social-based, demographics-based, etc.) can be combined.

Researchers in Machine Learning have known for long that the combination of classifiers usually achieves better results than each method separately, which is also true in Recommender Systems – the Netflix prize has been a paradigmatic example of this, where all the top classified teams used large recommender ensembles. We focus on weighted hybrid approaches, as an option that begets a simple and general formulation of the dynamic balance of the combined methods  $R_k$  by just setting the weights  $\lambda_k$  of each method in the hybrid combination. This approach can be expressed as follows:

$$\tilde{r}(u, i) = \sum_k \lambda_k * \tilde{r}_{R_k}(u, i) \text{ s.t. } \sum_k \lambda_k = 1 \quad (7.1)$$

In this chapter we investigate whether the performance predictors proposed in the previous chapter – where we have already found degrees of correlation between the ambiguity (clarity) of the user’s preferences and the accuracy of the system’s recommendations – can be useful for hybridisation. Specifically, we aim to use these predictors to build **dynamic hybrid recommenders** in such a way that the weight  $\lambda_k$  depends not only on the recommender but also on the current user  $u$ , or potentially other variables such as the item  $i$  or other available context information. We propose to specify such weights according to the ambiguity of the user’s preferences or item’s patterns, that is, we aim to use the performance predictors defined in the Chapter 6 to estimate those weights.

In the next section we propose a framework to perform dynamic hybrid recommendation where we use recommendation performance predictors and we analyse different requirements related to the adaptation of such predictors to produce weights in a hybrid recommender combination. After that, three different experiments are presented, where the predictors proposed in Chapter 6 are used as dynamic weights in the combination.

## 7.2 A performance prediction framework for ensemble recommendation

Let us simplify Equation (7.1) to the case where only two recommenders  $R1$  and  $R2$  are used. In this situation, only one weighting factor  $\lambda$  is needed (because of the constraint for the weights to sum to one) and we would have the following formulation:

$$\tilde{r}(u, i) = \lambda * \tilde{r}_{R1}(u, i) + (1 - \lambda) * \tilde{r}_{R2}(u, i) \quad (7.2)$$

In this case, since the  $\lambda$  weight is the same for every user  $u$  and item  $i$  we refer to such a recommender as a *static hybrid*. However, a single value of the combination parameter  $\lambda$  is not generally the optimal for each (user, item) pair. Therefore, instead of Equation (7.2), we may want to consider:

$$\tilde{r}(u, i) = \gamma_{R1}(u, i) * \tilde{r}_{R1}(u, i) + \gamma_{R2}(u, i) * \tilde{r}_{R2}(u, i) \quad (7.3)$$

where  $\gamma_R$  is the combination parameter which may depend on the current user, item, or both, and probably also depending on the recommender  $R$ . In this case we refer to such method as a *dynamic hybrid*.

A suitable assignment of the  $\gamma(u, i)$  parameters is a difficult task. In our approach, however, we propose to use the performance prediction methodology developed in the previous chapter, whenever the predictors show some correlation with the performance of a recommender. In this way, since we have some evidence that the performance predictors are able to estimate in advance the performance of a user in a user or item basis, we can use such estimations to weight accordingly the ratings predicted for a given user and item pair by each recommender.

In this context, it is not granted in general to obtain improvements whenever a performance predictor is used in a dynamic ensemble. We have to devise a set of conditions in which such predictors may be used; moreover, the ensemble problem has to be well defined, which is not always true as we shall show. Hence, we define a framework for dynamic hybrid recommendation based on recommendation performance predictors, characterised by some prerequisites, a specific normalisation strategy, and a weighting distribution among recommenders. In this framework, the weights  $\gamma_R$  are obtained by transformations of the values obtained by a performance predictor, in a similar way as the work presented in (Yom-Tov et al., 2005b) on rank aggregation in Information Retrieval, but in the context of Recommender Systems.

### 7.2.1 Requirements

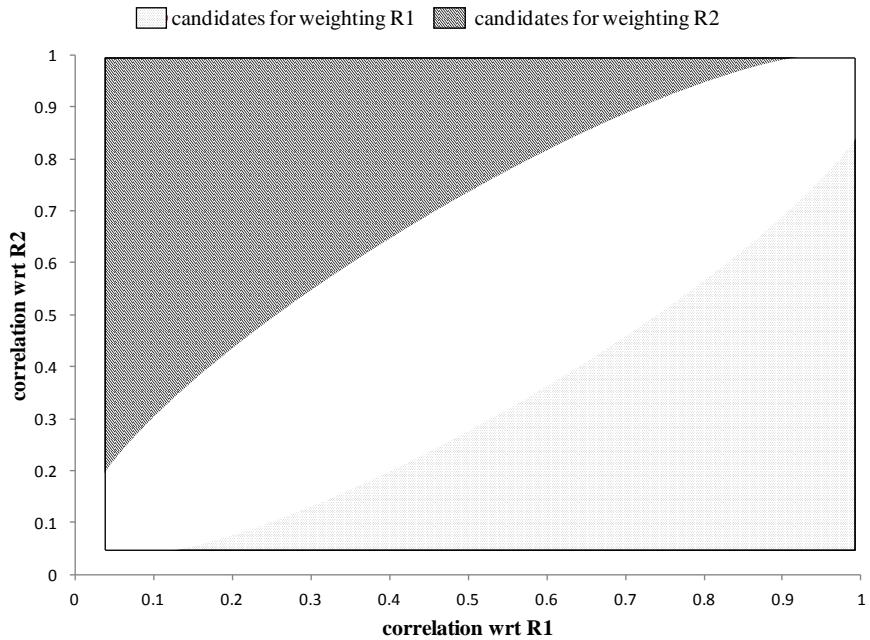
A first requirement to use a performance predictor for weighting the recommenders of an ensemble, is that it should correlate positively with the performance of not all

but some of such recommenders, or with the performance of all the recommenders but to different degrees. If a performance predictor correlates positively with all the recommenders in an ensemble to a similar extent, it does not provide a discriminative criteria to weight the recommenders any differently.

A predictor should be used to assign weights to those recommenders of the ensemble with which it correlates for performance. These assignments also alter the weights of the uncorrelated recommenders, since the weights of all the recommenders in the ensemble need to sum to 1. However, this should not affect the overall performance contribution of these recommenders, as the resulting weight should correspond randomly with their performance (hence the unpredicted recommenders' weight can be expected to change for good as much as for bad, whereas the weight of predicted recommenders should change more often for good).

Figure 7.1 shows which correlations can be considered valid according to the statements presented above, for an ensemble with two recommenders R1 and R2. The horizontal axis depicts the correlation with respect R1 and the vertical axis with R2. Hence, the dotted area represents those situations where a predictor's correlation for R1 is higher than for R2, and thus, the predictor should weight R1. Analogously, the striped area represents the candidate situations where the predictor should weight R2. Furthermore, when correlations with R1 and R2 are too similar (diagonal) no weighting assignment is preferred, and thus, if a predictor lies in the white area it should be used for weighting neither R1 nor R2 for the reasons described above.

Another requirement is that a recommender should not have an always superior or always inferior performance to those of the rest of the ensemble's recommenders. Otherwise the problem is distorted by the fact that the best weight is the one that gets closest to 0 for the recommenders that systematically perform worse (or 1 for the best), regardless of how excellent or terribly bad is the applied strategy, or the predictive power of the approach, since a biased predictor (either towards 0 or 1, depending on which recommender (the worst or the best) such predictor is weighting) would obtain very good results. This issue is recognised in (van Setten, 2005) where the author presents the situation where all recommenders produce item suggestions that are all too low or all too high with respect to the true user's preferences, and then the recommender ensemble is less accurate than the best individual recommender. In summary, underperforming recommenders are useless in an ensemble to begin with, or equivalently, the over performing one(s) should be used alone, and thus, there is no true weighting problem to solve.



**Figure 7.1. Valid predictor correlation regions for a recommender ensemble of size 2.**

### 7.2.2 Predictor normalisation

The output of a predictor is required to correlate with the performance of a recommender, but it is not necessarily by itself a good value for weighting the recommender in an ensemble, as already pointed out in (Hauff et al., 2009). In order to generate appropriate weights, the predictor output should be transformed by a monotonic function into values on a comparable scale, such as simply [0,1]. We shall call this transformation “normalisation.”

In this context, different transformations can be applied. Mapping the minimum value to 0 and the maximum to 1 is the simplest transformation, also known as *min-max* score normalisation (Renda and Straccia, 2003). Another common approach is to map (named *rank-sim* by Renda and Straccia, 2003) the predictor scores onto evenly distributed points in the [0,1], preserving their order. Min-max preserves the original predictor score distribution, while rank-sim maps it onto a uniform distribution. There is no obvious a priori reason to decide which case is preferable, to preserve the original distribution, or to equalise it somehow, and in fact more complex normalisation techniques could be used, like the one proposed in (Fernández et al., 2006b).

### 7.2.3 Weight distribution among recommenders

Once the predictor output has been normalised, it still needs a final adjustment to ensure, among other things, that the sum of the weights assigned to the ensemble’s

recommenders is 1. How this step is done depends, mainly, on how many recommenders are weighted by predictors, more specifically on whether all or only some of the combined recommenders are treated by performance predictors. Hence, we consider two options for the distribution of the weights among the recommenders:

- a) Only some of the recommenders in the ensemble are given dynamic weights. The rest of the recommenders receive the same weight, ensuring the weights of the ensemble's recommenders sum up to 1. This can be done in different ways:
  - Assigning a weight of 0.5 to the unpredicted recommenders, and dividing all weights by the total sum. This strategy is named as *fixed weight* or **FW**.
  - Assigning the dynamic weights to the corresponding recommenders, if we assume that their sum is  $\leq 1$ , then we divide 1 minus the sum of dynamic coefficients equally among the unpredicted recommenders. We denote this strategy as *one minus* or **OM**. If the sum is greater than 1, we have to divide by the total sum and normalise it by the total number of predictors.
- b) All recommenders are weighted using a specific predictor per recommender. This is not easy to grant in general, as there may not be predictors for all the recommenders combined. In case this option is taken, the weights can be simply normalised by the sum of weights.

Furthermore, if the output of each recommender has a different range, it would be necessary to apply an additional normalisation step to the recommender scores. The most usual strategies are the ones described in the previous section: score or rank normalisation (Renda and Straccia, 2003).

## 7.3 Experimental results

We next report experiments assessing the usefulness of the proposed predictors for adjusting the weights of a recommender ensemble, once their predictive power has been confirmed against the recommenders' actual performance, as reported in the previous chapter. We identify the combinations of recommenders that meet the conditions stated in the previous section for the dynamic combination problem to make sense and select the performance predictors to be applied based on their observed correlation with the performance of the recommenders (as reported in Section 6.5), and the requirements proposed in this chapter, i.e., that one recommender in the ensemble should have a positive correlation with the predictor, and the other should have an opposite or near neutral correlation. Then, we compare dynamic against static ensembles.

Among the different ways to set up static ensembles of two recommenders we take as baselines a) the best performing one in test, and b) the best theoretical static one without prior information, i.e., one with  $\lambda = 0.5$ . Intuitively, an even weighting

is the optimum over the – theoretical – set of all recommender ensembles: if say  $\lambda_B = 0.3$  was the best weight for the combination of two recommenders R1+R2, then  $\lambda = 0.3$  should be fairly bad for the permutation R2+R1 ( $\lambda = 1 - 0.3 = 0.7$  being best). If we assume that performance loss is convex with respect to  $|\lambda - \lambda_B|$  – it can be seen that otherwise the hybrid may underperform its constituents –, then  $\lambda = 0.5$  is the best compromise for R1+R2 and R2+R1. Since the set of all possible ensembles includes all the permutations of the combined recommenders,  $\lambda = 0.5$  is the best (theoretical) overall weight.

We also take as “skylines” (upper bound baselines) an oracle performance predictor consisting of the performance of the recommender itself. We shall refer to this method as ‘perfect correlation’, where the true performance of both recommenders is used as a weight for hybridisation (hence, such predictor would have a correlation of 1.0 with the recommender’s performance), whereas we shall refer to it as ‘PC-OM’ and ‘PC-FW’ when the performance of only one recommender is used (the same recommender being weighted by the predictors) along with the one minus or the fixed weight strategy for weight distribution (see Section 7.2.3). In all cases we apply a rank normalisation technique on the recommenders’ scores.

In the subsequent sections we present three experiments conducted to evaluate the proposed performance predictors. In the first experiment we use the rating-based predictors and test both user- and item-based performance predictors presented in Section 6.2.1. We use the MovieLens dataset, and compare the results with four of the evaluation methodologies presented in Chapter 4, i.e., AR, 1R, P1R, and U1R. In the second experiment we use predictors based on log data. We evaluate the predictors presented in Section 6.2.2 on the two versions of the Last.fm dataset using the 1R methodology. Finally, in the third experiment we test the social-based predictors presented in Section 6.3 on the CAMRa dataset and the AR methodology.

### 7.3.1 Dynamic recommender ensembles on rating data

As a first instantiation of our framework for building dynamic recommender ensembles described in Section 7.2, we first have to identify the recommenders to combine, that is: one of the recommenders should have a positive correlation with the predictor, while the other should have an opposite or near neutral correlation; besides, they should not perform very differently.

According to the correlation results presented in Section 6.5.1, we identify the pairs of recommenders presented in Table 7.1 as combinations meeting the conditions stated above. The first three ensembles are combinations of a collaborative filtering with a content-based recommendation method. The last ensemble combines a user-based collaborative filtering method with a non-personalised method, and the rest of the ensembles are combinations of two collaborative filtering methods. Al-

|      | R1   | R2      |
|------|------|---------|
| HRU1 | TFL1 | CB      |
| HRU2 | TFL2 | CB      |
| HRU3 | kNN  | CB      |
| HRU4 | kNN  | IB      |
| HRU5 | kNN  | pLSA    |
| HRU6 | kNN  | ItemPop |

**Table 7.1. Selected recommenders for building dynamic ensemble using user performance predictors that exploit rating-based information (MovieLens dataset).**

though some of these combinations have not been typical in the recommender systems literature, in our study they serve as a proof of concept to check whether the proposed dynamic recommender ensemble framework is useful in general or not. We refer the reader to Appendix A.2 for more details about the implementation of the recommenders.

The first two rows of Table 7.2, Table 7.3, Table 7.4, and Table 7.5 show the P@10 values for each of the combined recommenders obtained using the AR, 1R, U1R, and P1R methodologies, respectively. In Appendix A.5.1 we report results with other evaluation metrics. Note that, as mentioned in Chapter 4, in the AR methodology the absolute values are not meaningful since they depend on the amount of relevant information in test; on the other hand, for the 1R related methodologies (i.e., 1R, U1R, and P1R) the precision at 10 metric has an upper bound on 0.1, since there is only one relevant item in each ranking.

In these tables we may observe that among the six considered ensembles, there are cases where the first recommender (with respect to which the performance is predicted) performs better, worse, or similarly to the second recommender. This situation changes across methodologies and provides for a comparison of the resulting effects when the stated requirements are not met. Analogously, the predictors' correlations may change depending on the evaluation methodology followed, as observed in Section 6.5.1. Specifically, the recommenders presented in Table 7.1 were chosen according to the correlation results obtained for the AR methodology, and we may observe that some of the conditions stated above do not hold for some of the selected cases, for instance, correlation between most of the predictors and kNN recommender is negligible in the 1R, U1R, and P1R methodologies, in contrast with the results found for the AR methodology.

In the tables we may also observe that the best static ensemble is different depending on the evaluation methodology and the combined recommenders. The performance values of the best static ensembles, on the other hand, show an interesting situation that does depend on the specific considered ensemble, namely, whether the (best) static ensembles outperform or not both recommenders. For the AR methodology (Table 7.2), in the case of HRU1, HRU3, HRU5, and HRU6, the best static

|                                  | HRU1                             | HRU2                                | HRU3                                | HRU4                             | HRU5                                | HRU6                             |
|----------------------------------|----------------------------------|-------------------------------------|-------------------------------------|----------------------------------|-------------------------------------|----------------------------------|
| R1 ( $\lambda=1.0$ )             | 0.0024                           | 0.0696                              | 0.0307                              | 0.0307                           | 0.0307                              | 0.0307                           |
| R2 ( $\lambda=0.0$ )             | 0.0163                           | 0.0163                              | 0.0163                              | 0.0001                           | 0.1454                              | 0.0897                           |
| Baseline ( $\lambda=0.5$ )       | 0.0106                           | 0.0473                              | 0.0363                              | 0.0008                           | 0.1142                              | 0.0808                           |
| Best static<br>(best $\lambda$ ) | 0.0180<br>(0.1)                  | 0.0668<br>(0.9)                     | 0.0392<br>(0.9)                     | 0.0078<br>(0.9)                  | 0.1475<br>(0.1)                     | 0.0937<br>(0.1)                  |
| Perfect correlation              | <u>0.0189</u>                    | <u>0.0732</u>                       | <u>0.0401</u>                       | <u>0.0311</u>                    | 0.1469                              | 0.0980                           |
| PC-OM                            | 0.0176                           | 0.0721                              | 0.0434                              | 0.0091                           | 0.1489                              | 0.0958                           |
| PC-FW                            | 0.0177                           | 0.0541                              | 0.0379                              | 0.0025                           | 0.1478                              | 0.0958                           |
| Entropy-OM                       | <b>0.0110<math>\nabla</math></b> | <b>0.0685<math>\triangle</math></b> | <b>0.0388<math>\nabla</math></b>    | <b>0.0069<math>\nabla</math></b> | 0.1126 $\nabla$                     | 0.0791 $\nabla$                  |
| ItemSimple-OM                    | <b>0.0170<math>\nabla</math></b> | <b>0.0685<math>\triangle</math></b> | <b>0.0390<math>\nabla</math></b>    | <b>0.0072<math>\nabla</math></b> | <b>0.1496<math>\triangle</math></b> | <b>0.0919<math>\nabla</math></b> |
| ItemUser-OM                      | <b>0.0172<math>\nabla</math></b> | <b>0.0680<math>\triangle</math></b> | <b>0.0386<math>\nabla</math></b>    | <b>0.0068<math>\nabla</math></b> | <b>0.1513<math>\triangle</math></b> | <b>0.0924<math>\nabla</math></b> |
| RatUser-OM                       | <b>0.0177<math>\nabla</math></b> | <b>0.0687<math>\triangle</math></b> | <b>0.0393<math>\triangle</math></b> | <b>0.0072<math>\nabla</math></b> | <b>0.1535<math>\triangle</math></b> | <b>0.0931<math>\nabla</math></b> |
| RatItem-OM                       | <b>0.0178<math>\nabla</math></b> | <b>0.0674<math>\triangle</math></b> | <b>0.0389<math>\nabla</math></b>    | <b>0.0066<math>\nabla</math></b> | <b>0.1542<math>\triangle</math></b> | <b>0.0928<math>\nabla</math></b> |
| IRUser-OM                        | <b>0.0169<math>\nabla</math></b> | <b>0.0668<math>\triangle</math></b> | <b>0.0387<math>\nabla</math></b>    | <b>0.0066<math>\nabla</math></b> | <b>0.1487<math>\triangle</math></b> | <b>0.0922<math>\nabla</math></b> |
| IRItem-OM                        | <b>0.0172<math>\nabla</math></b> | <b>0.0655<math>\nabla</math></b>    | <b>0.0378<math>\nabla</math></b>    | <b>0.0061<math>\nabla</math></b> | <b>0.1500<math>\triangle</math></b> | <b>0.0918<math>\nabla</math></b> |
| IRUserItem-OM                    | <b>0.0170<math>\nabla</math></b> | <b>0.0665<math>\nabla</math></b>    | <b>0.0388<math>\nabla</math></b>    | <b>0.0066<math>\nabla</math></b> | <b>0.1498<math>\triangle</math></b> | <b>0.0916<math>\nabla</math></b> |
| Entropy-FW                       | <b>0.0111<math>\nabla</math></b> | <b>0.0528<math>\nabla</math></b>    | <b>0.0369<math>\nabla</math></b>    | <b>0.0027<math>\nabla</math></b> | <b>0.1156<math>\nabla</math></b>    | 0.0807 $\nabla$                  |
| ItemSimple-FW                    | <b>0.0156<math>\nabla</math></b> | <b>0.0529<math>\nabla</math></b>    | <b>0.0369<math>\nabla</math></b>    | <b>0.0027<math>\nabla</math></b> | <b>0.1433<math>\nabla</math></b>    | <b>0.0908<math>\nabla</math></b> |
| ItemUser-FW                      | <b>0.0166<math>\nabla</math></b> | <b>0.0529<math>\nabla</math></b>    | <b>0.0368<math>\nabla</math></b>    | <b>0.0028<math>\nabla</math></b> | <b>0.1468<math>\nabla</math></b>    | <b>0.0915<math>\nabla</math></b> |
| RatUser-FW                       | <b>0.0170<math>\nabla</math></b> | <b>0.0528<math>\nabla</math></b>    | <b>0.0370<math>\nabla</math></b>    | <b>0.0028<math>\nabla</math></b> | <b>0.1498<math>\triangle</math></b> | <b>0.0919<math>\nabla</math></b> |
| RatItem-FW                       | <b>0.0170<math>\nabla</math></b> | <b>0.0529<math>\nabla</math></b>    | <b>0.0369<math>\nabla</math></b>    | <b>0.0027<math>\nabla</math></b> | <b>0.1499<math>\triangle</math></b> | <b>0.0918<math>\nabla</math></b> |
| IRUser-FW                        | <b>0.0161<math>\nabla</math></b> | <b>0.0526<math>\nabla</math></b>    | <b>0.0371<math>\nabla</math></b>    | <b>0.0029<math>\nabla</math></b> | <b>0.1420<math>\nabla</math></b>    | <b>0.0912<math>\nabla</math></b> |
| IRItem-FW                        | <b>0.0163<math>\nabla</math></b> | <b>0.0525<math>\nabla</math></b>    | <b>0.0367<math>\nabla</math></b>    | <b>0.0027<math>\nabla</math></b> | <b>0.1459<math>\nabla</math></b>    | <b>0.0909<math>\nabla</math></b> |
| IRUserItem-FW                    | <b>0.0164<math>\nabla</math></b> | <b>0.0527<math>\nabla</math></b>    | <b>0.0372<math>\nabla</math></b>    | <b>0.0028<math>\nabla</math></b> | <b>0.1452<math>\nabla</math></b>    | <b>0.0908<math>\nabla</math></b> |

**Table 7.2. Dynamic ensemble performance values (P@10) using AR methodology and user predictors (MovieLens dataset).** Improvements over the baseline are in bold, the best result for each column is underlined. The value  $a$  of each dynamic hybrid is marked with  $a_y^x$ , where  $x$  and  $y$  indicate, respectively, statistical difference with respect to the best static (upper,  $x$ ) and with respect to the baseline (lower,  $y$ ). Moreover,  $\blacktriangle$  and  $\triangle$  indicate, respectively, significant and non-significant improvements over the corresponding recommender. A similar convention with  $\blacktriangledown$  and  $\nabla$  indicates values below the recommender performance. Statistical significance is established by paired Wilcoxon  $p < 0.05$  in all cases.

outperforms both recommenders, but this is not observed for HRU2 nor for HRU4. In the latter scenarios, thus, it seems hybridisation would not be so useful for combination.

Additionally, regarding the normalisation of the predictor's output we evaluate two normalisation techniques: rank and score normalisation. Since there is no prior information about which normalisation technique would provide better results, we test both, and report the best results in each situation, which are usually achieved by the rank-sim normalisation technique. Finally, the weigh strategy is also included as a parameter of the experiments. Since we only have a predictor for one of the recommenders in the ensemble (denoted as R1), as we explained in Section 7.2.3, we may weight the unpredicted recommender as one minus the predictor value (OM), or as 0.5 and then divide the weights of the two recommenders by the sum of weights (FW).

|                                  | HRU1                                | HRU2  | HRU3  | HRU4  | HRU5   | HRU6   |
|----------------------------------|-------------------------------------|---|---|---|--|--|
| R1 ( $\lambda=1.0$ )             | 0.0221                              | 0.0690  | 0.0437  | 0.0437  | 0.0437                                       | 0.0437                                       |
| R2 ( $\lambda=0.0$ )             | 0.0221                              | 0.0221  | 0.0221  | 0.0074  | 0.0836                                       | 0.0649                                       |
| Baseline ( $\lambda=0.5$ )       | 0.0338                              | 0.0536  | 0.0469  | 0.0327  | 0.0749                                       | 0.0658                                       |
| Best static<br>(best $\lambda$ ) | 0.0338<br>(0.4)                     | <u>0.0720</u><br>(0.9)                        | 0.0514<br>(0.8)                               | 0.0455<br>(0.9)                               | <u>0.0856</u><br>(0.1)                       | 0.0696<br>(0.2)                              |
| Perfect correlation              | <u>0.0370</u>                       | 0.0715  | <u>0.0553</u>                                 | <u>0.0458</u>                                 | 0.0840                                       | <u>0.0723</u>                                |
| PC-OM                            | 0.0358                              | 0.0683  | 0.0507  | 0.0353  | 0.0811                                       | 0.0709                                       |
| PC-FW                            | 0.0343                              | 0.0592  | 0.0482  | 0.0344  | 0.0803                                       | 0.0699                                       |
| Entropy-OM                       | 0.0332 $\downarrow$                 | <b>0.0662<math>\blacktriangleleft</math></b>  | <b>0.0472<math>\triangle</math></b>           | <b>0.0382<math>\blacktriangleright</math></b> | 0.0709 $\downarrow$                          | 0.0626 $\downarrow$                          |
| ItemSimple-OM                    | 0.0304 $\downarrow$                 | <b>0.0666<math>\blacktriangleleft</math></b>  | <b>0.0473<math>\blacktriangleleft</math></b>  | <b>0.0384<math>\blacktriangleright</math></b> | <b>0.0844<math>\blacktriangleleft</math></b> | <b>0.0681<math>\blacktriangleleft</math></b> |
| ItemUser-OM                      | 0.0305 $\downarrow$                 | <b>0.0660<math>\blacktriangleleft</math></b>  | <b>0.0471<math>\blacktriangleright</math></b> | <b>0.0381<math>\blacktriangleright</math></b> | <b>0.0847<math>\blacktriangleleft</math></b> | <b>0.0680<math>\blacktriangleleft</math></b> |
| RatUser-OM                       | 0.0307 $\downarrow$                 | <b>0.0666<math>\blacktriangleleft</math></b>  | <b>0.0478<math>\blacktriangleright</math></b> | <b>0.0386<math>\blacktriangleright</math></b> | <b>0.0850<math>\blacktriangleleft</math></b> | <b>0.0680<math>\blacktriangleleft</math></b> |
| RatItem-OM                       | 0.0305 $\downarrow$                 | <b>0.0663<math>\blacktriangleleft</math></b>  | <b>0.0475<math>\blacktriangleright</math></b> | <b>0.0385<math>\blacktriangleright</math></b> | <b>0.0849<math>\blacktriangleleft</math></b> | <b>0.0678<math>\blacktriangleleft</math></b> |
| IRUser-OM                        | 0.0304 $\downarrow$                 | <b>0.0655<math>\blacktriangleleft</math></b>  | <b>0.0470<math>\triangle</math></b>           | <b>0.0381<math>\blacktriangleright</math></b> | <b>0.0839<math>\blacktriangleleft</math></b> | <b>0.0675<math>\blacktriangleleft</math></b> |
| IRItem-OM                        | 0.0298 $\downarrow$                 | <b>0.0644<math>\blacktriangleleft</math></b>  | 0.0457 $\downarrow$                           | <b>0.0370<math>\blacktriangleright</math></b> | <b>0.0839<math>\blacktriangleleft</math></b> | <b>0.0671<math>\blacktriangleleft</math></b> |
| IRUserItem-OM                    | 0.0305 $\downarrow$                 | <b>0.0655<math>\blacktriangleleft</math></b>  | <b>0.0471<math>\triangle</math></b>           | <b>0.0381<math>\blacktriangleright</math></b> | <b>0.0841<math>\blacktriangleleft</math></b> | <b>0.0674<math>\blacktriangleleft</math></b> |
| Entropy-FW                       | <b>0.0339<math>\triangle</math></b> | <b>0.0594<math>\blacktriangleright</math></b> | <b>0.0472<math>\triangle</math></b>           | <b>0.0356<math>\blacktriangleright</math></b> | 0.0686 $\downarrow$                          | 0.0650 $\downarrow$                          |
| ItemSimple-FW                    | 0.0321 $\downarrow$                 | <b>0.0596<math>\blacktriangleright</math></b> | <b>0.0473<math>\blacktriangleright</math></b> | <b>0.0358<math>\blacktriangleright</math></b> | <b>0.0837<math>\blacktriangleleft</math></b> | <b>0.0684<math>\blacktriangleleft</math></b> |
| ItemUser-FW                      | 0.0320 $\downarrow$                 | <b>0.0594<math>\blacktriangleright</math></b> | <b>0.0471<math>\triangle</math></b>           | <b>0.0356<math>\blacktriangleright</math></b> | <b>0.0843<math>\blacktriangleleft</math></b> | <b>0.0683<math>\blacktriangleleft</math></b> |
| RatUser-FW                       | 0.0321 $\downarrow$                 | <b>0.0596<math>\blacktriangleright</math></b> | <b>0.0475<math>\blacktriangleright</math></b> | <b>0.0359<math>\blacktriangleright</math></b> | <b>0.0848<math>\blacktriangleleft</math></b> | <b>0.0684<math>\blacktriangleleft</math></b> |
| RatItem-FW                       | 0.0321 $\downarrow$                 | <b>0.0595<math>\blacktriangleright</math></b> | <b>0.0473<math>\blacktriangleright</math></b> | <b>0.0358<math>\blacktriangleright</math></b> | <b>0.0847<math>\blacktriangleleft</math></b> | <b>0.0684<math>\blacktriangleleft</math></b> |
| IRUser-FW                        | 0.0320 $\downarrow$                 | <b>0.0592<math>\blacktriangleright</math></b> | <b>0.0471<math>\triangle</math></b>           | <b>0.0356<math>\blacktriangleright</math></b> | <b>0.0834<math>\blacktriangleleft</math></b> | <b>0.0680<math>\blacktriangleleft</math></b> |
| IRItem-FW                        | 0.0318 $\downarrow$                 | <b>0.0588<math>\blacktriangleright</math></b> | 0.0465 $\downarrow$                           | <b>0.0349<math>\blacktriangleright</math></b> | <b>0.0835<math>\blacktriangleleft</math></b> | <b>0.0674<math>\blacktriangleleft</math></b> |
| IRUserItem-FW                    | 0.0320 $\downarrow$                 | <b>0.0592<math>\blacktriangleright</math></b> | <b>0.0471<math>\triangle</math></b>           | <b>0.0356<math>\blacktriangleright</math></b> | <b>0.0837<math>\blacktriangleleft</math></b> | <b>0.0678<math>\blacktriangleleft</math></b> |

**Table 7.3. Dynamic ensemble performance values (P@10) using 1R methodology and user predictors (MovieLens dataset).**

Table 7.2 shows the results obtained following the AR methodology. We may observe how, except in three cases, dynamic ensembles outperform the baseline. Interestingly, for HRU5, the best performing method is not the one obtained with the ‘perfect correlation’ approach, as we may expect, but with our dynamic ensembles based on the user clarity performance predictors. This is due to the fact that the corresponding predictor for the first recommender (P@10 values for kNN) also has a strong correlation with the performance of the second recommender (pLSA), and thus, it does not satisfy the requirement that the correlation values should not be too similar for both recommenders.

Table 7.3 shows the results obtained with the 1R methodology. Note that in this case the correlations were consistently lower than those obtained with the AR methodology. In particular, this is emphasised in the results of the dynamic ensemble HRU1, which do not outperform the baseline for almost any predictor. This can be explained with the results reported in Table 6.9, where the TFL1 recommender obtains a near-zero correlation, and thus, the correlation requirement of our framework is not satisfied. Specifically, this fact highlights the importance of the strength in the correlation between the predictor and the recommender performance, as stated in Section 7.2.1. Furthermore, we may observe in the table that for two combinations

|                                  | HRU1            | HRU2            | HRU3            | HRU4            | HRU5            | HRU6            |
|----------------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| R1 ( $\lambda=1.0$ )             | 0.0294          | 0.0524          | 0.0381          | 0.0381          | 0.0381          | 0.0381          |
| R2 ( $\lambda=0.0$ )             | 0.0223          | 0.0223          | 0.0223          | 0.0068          | 0.0718          | 0.0406          |
| Baseline ( $\lambda=0.5$ )       | 0.0345          | 0.0440          | 0.0396          | 0.0283          | 0.0639          | 0.0493          |
| Best static<br>(best $\lambda$ ) | 0.0351<br>(0.6) | 0.0536<br>(0.9) | 0.0424<br>(0.7) | 0.0384<br>(0.9) | 0.0732<br>(0.1) | 0.0493<br>(0.5) |
| Perfect correlation              | <b>0.0389</b>   | <b>0.0552</b>   | <b>0.0493</b>   | <b>0.0396</b>   | <b>0.0742</b>   | <b>0.0559</b>   |
| PC-OM                            | 0.0373          | 0.0485          | 0.0471          | 0.0332          | 0.0732          | 0.0548          |
| PC-FW                            | 0.0355          | 0.0459          | 0.0429          | 0.0307          | 0.0722          | 0.0535          |
| Entropy-OM                       | 0.0345▼         | <b>0.0518▼</b>  | <b>0.0404▼</b>  | <b>0.0337▼</b>  | 0.0615▼         | 0.0471▼         |
| ItemSimple-OM                    | 0.0333▼         | <b>0.0519▼</b>  | <b>0.0403▼</b>  | <b>0.0339▼</b>  | <b>0.0723▼</b>  | 0.0444▼         |
| ItemUser-OM                      | 0.0334▼         | <b>0.0517▼</b>  | <b>0.0403▼</b>  | <b>0.0336▼</b>  | <b>0.0726▼</b>  | 0.0438▼         |
| RatUser-OM                       | 0.0335▼         | <b>0.0521▼</b>  | <b>0.0410▼</b>  | <b>0.0341▼</b>  | <b>0.0728▼</b>  | 0.0435▼         |
| RatItem-OM                       | 0.0334▼         | <b>0.0516▼</b>  | <b>0.0406▼</b>  | <b>0.0341▼</b>  | <b>0.0726▼</b>  | 0.0434▼         |
| IRUser-OM                        | 0.0333▼         | <b>0.0511▼</b>  | <b>0.0401▼</b>  | <b>0.0336▼</b>  | <b>0.0718▼</b>  | 0.0440▼         |
| IRItem-OM                        | 0.0326▼         | <b>0.0504▼</b>  | 0.0388▼         | <b>0.0325▼</b>  | <b>0.0714▼</b>  | 0.0430▼         |
| IRUserItem-OM                    | 0.0334▼         | <b>0.0511▼</b>  | <b>0.0401▼</b>  | <b>0.0336▼</b>  | <b>0.0719▼</b>  | 0.0437▼         |
| Entropy-FW                       | <b>0.0347▼</b>  | <b>0.0472▼</b>  | <b>0.0402▼</b>  | <b>0.0308▼</b>  | 0.0636▼         | 0.0486▼         |
| ItemSimple-FW                    | 0.0342▼         | <b>0.0473▼</b>  | <b>0.0402▼</b>  | <b>0.0309▼</b>  | <b>0.0720▼</b>  | 0.0467▼         |
| ItemUser-FW                      | 0.0342▼         | <b>0.0471▼</b>  | <b>0.0401▼</b>  | <b>0.0308▼</b>  | <b>0.0724▼</b>  | 0.0467▼         |
| RatUser-FW                       | 0.0343▼         | <b>0.0474▼</b>  | <b>0.0405▼</b>  | <b>0.0310▼</b>  | <b>0.0727▼</b>  | 0.0469▼         |
| RatItem-FW                       | 0.0342▼         | <b>0.0472▼</b>  | <b>0.0403▼</b>  | <b>0.0309▼</b>  | <b>0.0725▼</b>  | 0.0469▼         |
| IRUser-FW                        | 0.0341▼         | <b>0.0470▼</b>  | <b>0.0401▼</b>  | <b>0.0308▼</b>  | <b>0.0714▼</b>  | 0.0469▼         |
| IRItem-FW                        | 0.0338▼         | <b>0.0467▼</b>  | 0.0393▼         | <b>0.0302▼</b>  | <b>0.0712▼</b>  | 0.0464▼         |
| IRUserItem-FW                    | 0.0341▼         | <b>0.0471▼</b>  | <b>0.0401▼</b>  | <b>0.0308▼</b>  | <b>0.0716▼</b>  | 0.0469▼         |

**Table 7.4. Dynamic ensemble performance values (P@10) using the U1R methodology and user predictors (MovieLens dataset)**

(HRU2 and HRU5) the best performance results are not obtained by dynamic approaches, but by the best static approaches in contrast with what we found for the AR methodology. This situation is different to the one obtained when we evaluate using MAP@10 (see Appendix A.4.1), where the best results are always obtained by dynamic ensembles.

Table 7.4 and Table 7.5 show the performance values obtained with the unbiased methodologies proposed in Chapter 4, that is, U1R and P1R. Following the U1R methodology (Table 7.4) we obtain similar results to those obtained in the 1R methodology except for HRU6. In contrast, with the P1R methodology (Table 7.5) our framework does not show improvements over any baseline. We may see that the ‘perfect correlation’ methods are able to obtain better, although very close, values than those of the best static ensemble. This means that there is room for improvement in this methodology, and that the performance of the dynamic recommender ensembles could be improved if better performance predictors were found.

|                                  | HRU1                   | HRU2                   | HRU3            | HRU4            | HRU5                   | HRU6            |
|----------------------------------|------------------------|------------------------|-----------------|-----------------|------------------------|-----------------|
| R1 ( $\lambda=1.0$ )             | 0.0203                 | 0.0348                 | 0.0265          | 0.0265          | 0.0265                 | 0.0265          |
| R2 ( $\lambda=0.0$ )             | 0.0197                 | 0.0197                 | 0.0197          | 0.0208          | 0.0604                 | 0.0282          |
| Baseline ( $\lambda=0.5$ )       | <u>0.0470</u>          | 0.0579                 | 0.0539          | 0.0269          | 0.0763                 | 0.0560          |
| Best static<br>(best $\lambda$ ) | <u>0.0470</u><br>(0.5) | <u>0.0593</u><br>(0.6) | 0.0541<br>(0.6) | 0.0278<br>(0.7) | <u>0.0796</u><br>(0.4) | 0.0560<br>(0.5) |
| Perfect correlation              | 0.0464                 | 0.0579                 | <u>0.0546</u>   | <u>0.0314</u>   | 0.0767                 | <u>0.0564</u>   |
| PC-OM                            | 0.0425                 | 0.0554                 | 0.0528          | 0.0296          | 0.0746                 | <u>0.0537</u>   |
| PC-FW                            | 0.0429                 | 0.0542                 | 0.0504          | 0.0282          | 0.0764                 | 0.0522          |
| Entropy-OM                       | 0.0431▼                | 0.0564▼                | 0.0502▼         | 0.0261▼         | 0.0698▼                | 0.0521▼         |
| ItemSimple-OM                    | 0.0358▼                | 0.0509▼                | 0.0429▼         | 0.0261▼         | 0.0689▼                | 0.0441▼         |
| ItemUser-OM                      | 0.0361▼                | 0.0512▼                | 0.0431▼         | 0.0261▼         | 0.0675▼                | 0.0444▼         |
| RatUser-OM                       | 0.0362▼                | 0.0514▼                | 0.0436▼         | 0.0263▼         | 0.0663▼                | 0.0446▼         |
| RatItem-OM                       | 0.0361▼                | 0.0511▼                | 0.0432▼         | 0.0262▼         | 0.0661▼                | 0.0444▼         |
| IRUser-OM                        | 0.0365▼                | 0.0513▼                | 0.0435▼         | 0.0263▼         | 0.0687▼                | 0.0447▼         |
| IRItem-OM                        | 0.0357▼                | 0.0504▼                | 0.0421▼         | 0.0257▼         | 0.0669▼                | 0.0439▼         |
| IRUserItem-OM                    | 0.0365▼                | 0.0513▼                | 0.0434▼         | 0.0263▼         | 0.0675▼                | 0.0447▼         |
| Entropy-FW                       | 0.0457▼                | 0.0577▼                | 0.0524▼         | 0.0265▼         | 0.0745▼                | 0.0546▼         |
| ItemSimple-FW                    | 0.0410▼                | 0.0540▼                | 0.0475▼         | 0.0266▼         | 0.0720▼                | 0.0498▼         |
| ItemUser-FW                      | 0.0409▼                | 0.0538▼                | 0.0473▼         | 0.0265▼         | 0.0706▼                | 0.0497▼         |
| RatUser-FW                       | 0.0410▼                | 0.0540▼                | 0.0477▼         | 0.0267▼         | 0.0691▼                | 0.0499▼         |
| RatItem-FW                       | 0.0411▼                | 0.0541▼                | 0.0476▼         | 0.0266▼         | 0.0688▼                | 0.0499▼         |
| IRUser-FW                        | 0.0410▼                | 0.0538▼                | 0.0474▼         | 0.0266▼         | 0.0721▼                | 0.0496▼         |
| IRItem-FW                        | 0.0406▼                | 0.0534▼                | 0.0467▼         | 0.0263▼         | 0.0699▼                | 0.0491▼         |
| IRUserItem-FW                    | 0.0409▼                | 0.0538▼                | 0.0474▼         | 0.0266▼         | 0.0706▼                | 0.0496▼         |

**Table 7.5. Dynamic ensemble performance values (P@10) using the P1R methodology and user predictors (MovieLens dataset).**

In summary, the **results show that our methods significantly outperform static ensembles for different recommender combinations in most of the evaluation methodologies**. Moreover, in most cases our methods also achieve the best results for each ensemble, let aside the performance of the oracle performance prediction (perfect correlation) and best static approaches, which use groundtruth (test) information, differently to the clarity- and entropy-based performance predictors.

Nevertheless, we observe that in those cases where the dynamic ensembles do not perform better than the static ensembles, the best static approaches use values of  $\lambda$  close to 0.5. We hypothesise that our framework may be biased towards favouring those ensembles whose recommender combination is highly unbalanced. Interestingly, although the predictors only weight one of the recommenders (not always the better performing one) a dynamic ensemble is usually able to find the optimal combination in the unbalanced cases. In particular, this could help to answer why our dynamic ensembles underperform static approaches for the U1R and P1R methodologies, since the best static in these cases seem to be often very close to 0.5.

|      | R1      | R2  |
|------|---------|-----|
| HRI1 | pLSA    | CB  |
| HRI2 | pLSA    | kNN |
| HRI3 | ItemPop | CB  |
| HRI4 | ItemPop | kNN |

**Table 7.6. Selected recommenders for building dynamic ensembles using item predictors that exploit rating data (MovieLens dataset).**

### Using item-based predictors

As we noted in Section 6.5.2, item-based predictors could also be valuable since they also obtain high correlations with respect to item performance. Table 7.6 shows the selected recommenders that satisfy the correlation requirements with item predictors. Table 7.7, Table 7.8, and Table 7.9 show the results obtained when these recommender combinations are evaluated and compared against dynamic versions (using our proposed item predictors), and using the 1R, U1R, and uuU1R methodologies. In this case, ensemble predictions are computed by means of Equation (7.3) with values  $\gamma(u, i)$  only depending on the current item, that is,  $\gamma(i)$ .

When measuring the performance of dynamic ensembles that use item-based performance predictors, we do not compute the perfect correlation predictors because we do not have a standard metric for item performance. Apart from that, the

|                                  | HRI1           | HRI2           | HRI3           | HRI4           |
|----------------------------------|----------------|----------------|----------------|----------------|
| R1 ( $\lambda=1.0$ )             | 0.0836         | 0.0836         | 0.0649         | 0.0649         |
| R2 ( $\lambda=0.0$ )             | 0.0221         | 0.0437         | 0.0221         | 0.0437         |
| Baseline ( $\lambda=0.5$ )       | 0.0909         | 0.0924         | 0.0886         | 0.0907         |
| Best static<br>(best $\lambda$ ) | 0.0909         | 0.0924         | 0.0886         | 0.0907         |
| Entropy-OM                       | 0.0708▼        | 0.0858▼        | 0.0684▼        | 0.0831▼        |
| UserSimple-OM                    | 0.0761▼        | 0.0905▼        | 0.0723▼        | 0.0837▼        |
| UserItem-OM                      | 0.0776▼        | 0.0903▼        | 0.0749▼        | 0.0843▼        |
| RatItem-OM                       | 0.0751▼        | 0.0893▼        | 0.0712▼        | 0.0824▼        |
| RatUser-OM                       | 0.0759▼        | 0.0892▼        | 0.0674▼        | 0.0789▼        |
| URIItem-OM                       | 0.0776▼        | 0.0911▼        | 0.0797▼        | 0.0885▼        |
| URUser-OM                        | 0.0781▼        | 0.0906▼        | 0.0721▼        | 0.0820▼        |
| URIItemUser-OM                   | 0.0777▼        | 0.0909▼        | 0.0777▼        | 0.0869▼        |
| Entropy-FW                       | 0.0798▼        | 0.0923▼        | 0.0771▼        | 0.0895▼        |
| UserSimple-FW                    | <b>0.0946▲</b> | <b>0.0979▲</b> | <b>0.0916▲</b> | <b>0.0949▲</b> |
| UserItem-FW                      | <b>0.0949▲</b> | <b>0.0980▲</b> | <b>0.0920▲</b> | <b>0.0950▲</b> |
| RatItem-FW                       | <b>0.0944▲</b> | <b>0.0979▲</b> | <b>0.0913▲</b> | <b>0.0948▲</b> |
| RatUser-FW                       | <b>0.0946▲</b> | <b>0.0978▲</b> | <b>0.0908▲</b> | <b>0.0942▲</b> |
| URIItem-FW                       | <b>0.0940▲</b> | <b>0.0981▲</b> | <b>0.0923▲</b> | <b>0.0958▲</b> |
| URUser-FW                        | <b>0.0946▲</b> | <b>0.0978▲</b> | <b>0.0912▲</b> | <b>0.0945▲</b> |
| URIItemUser-FW                   | <b>0.0944▲</b> | <b>0.0980▲</b> | <b>0.0921▲</b> | <b>0.0954▲</b> |

**Table 7.7. Dynamic ensemble performance values (P@10) using 1R methodology with item predictors (MovieLens dataset).**

rest of the experimental settings is the same as those described above for dynamic hybrids with user-based performance predictors.

Table 7.7 shows the results obtained by using item-based predictors and the 1R methodology. We may observe that if the predictors are weighted using the FW strategy, dynamic ensembles outperform static combinations in every situation, except for the Entropy predictor. It is interesting to note that, differently to user-based predictors, the dynamic ensembles are able to outperform the best static ensemble even when they are close to the baseline with  $\lambda = 0.5$ . The reader may compare Table 7.4 and Table 7.7 to observe these differences.

In Table 7.8, where the methodology U1R is used, a very similar situation occurs, although not all dynamic ensembles outperform the static approach with the FW strategy. Specifically, the dynamic hybrid weighted by the URItem clarity predictor clearly obtains better performance than the rest of the dynamic and static ensembles, in particular the HRI3 and HRI4 combinations.

Finally, the performance results found for the uuU1R methodology are presented in Table 7.9, in which the test ratings – i.e., the users – are uniformly distributed over the items, items previously uniformly distributed in the test (like in the U1R methodology). In this experiment, the performance of the dynamic ensemble is much better than in the previous experiments, since **all the rating-based item predictors (except for the Entropy predictor) outperform the static baseline no matter the weighting strategy in three out of four recommender combinations**.

|                                  | HRI1            | HRI2            | HRI3            | HRI4            |
|----------------------------------|-----------------|-----------------|-----------------|-----------------|
| R1 ( $\lambda=1.0$ )             | 0.0718          | 0.0718          | 0.0406          | 0.0406          |
| R2 ( $\lambda=0.0$ )             | 0.0223          | 0.0381          | 0.0223          | 0.0381          |
| Baseline ( $\lambda=0.5$ )       | 0.0764          | 0.0812          | 0.0630          | 0.0689          |
| Best static<br>(best $\lambda$ ) | 0.0764<br>(0.5) | 0.0812<br>(0.5) | 0.0630<br>(0.5) | 0.0689<br>(0.5) |
| Entropy-OM                       | 0.0571▼         | 0.0652▼         | 0.0435▼         | 0.0508▼         |
| UserSimple-OM                    | 0.0657▼         | 0.0716▼         | 0.0399▼         | 0.0450▼         |
| UserItem-OM                      | 0.0671▼         | 0.0721▼         | 0.0425▼         | 0.0462▼         |
| RatItem-OM                       | 0.0645▼         | 0.0699▼         | 0.0392▼         | 0.0435▼         |
| RatUser-OM                       | 0.0620▼         | 0.0671▼         | 0.0335▼         | 0.0382▼         |
| URItem-OM                        | 0.0705▼         | 0.0757▼         | 0.0496▼         | 0.0532▼         |
| URUser-OM                        | 0.0650▼         | 0.0699▼         | 0.0372▼         | 0.0414▼         |
| URIItemUser-OM                   | 0.0690▼         | 0.0741▼         | 0.0462▼         | 0.0500▼         |
| Entropy-FW                       | 0.0668▼         | 0.0757▼         | 0.0518▼         | 0.0595▼         |
| UserSimple-FW                    | <b>0.0840▲</b>  | <b>0.0886▲</b>  | 0.0601▼         | 0.0658▼         |
| UserItem-FW                      | <b>0.0844▲</b>  | <b>0.0887▲</b>  | 0.0609▼         | 0.0663▼         |
| RatItem-FW                       | <b>0.0839▲</b>  | <b>0.0883▲</b>  | 0.0598▼         | 0.0653▼         |
| RatUser-FW                       | <b>0.0831▲</b>  | <b>0.0876▲</b>  | 0.0573▼         | 0.0630▼         |
| URItem-FW                        | <b>0.0851▲</b>  | <b>0.0897▲</b>  | <b>0.0642▲</b>  | <b>0.0698▲</b>  |
| URUser-FW                        | <b>0.0836▲</b>  | <b>0.0881▲</b>  | 0.0585▼         | 0.0642▼         |
| URIItemUser-FW                   | <b>0.0848▲</b>  | <b>0.0893▲</b>  | 0.0625▼         | 0.0680▼         |

**Table 7.8. Dynamic ensemble performance values (P@10) using U1R methodology with item predictors (MovieLens dataset).**

|                            | HRI1           | HRI2           | HRI3           | HRI4           |
|----------------------------|----------------|----------------|----------------|----------------|
| R1 ( $\lambda=1.0$ )       | <u>0.0536</u>  | 0.0536         | 0.0225         | 0.0225         |
| R2 ( $\lambda=0.0$ )       | 0.0198         | 0.0275         | 0.0198         | 0.0275         |
| Baseline ( $\lambda=0.5$ ) | 0.0374         | 0.0440         | 0.0239         | 0.0256         |
| Best static                | 0.0491         | 0.0502         | 0.0239         | 0.0271         |
| (best $\lambda$ )          | (0.9)          | (0.9)          | (0.6)          | (0.2)          |
| Entropy-OM                 | 0.0324▼        | 0.0385▼        | 0.0236▼        | <b>0.0280▲</b> |
| UserSimple-OM              | <b>0.0510△</b> | <b>0.0548▲</b> | 0.0237▼        | <b>0.0282▲</b> |
| UserItem-OM                | <b>0.0514▲</b> | <b>0.0547▲</b> | 0.0236▼        | <b>0.0280▲</b> |
| RatItem-OM                 | <b>0.0516▲</b> | <b>0.0547▲</b> | 0.0237▼        | <b>0.0281▲</b> |
| RatUser-OM                 | <b>0.0523▲</b> | <b>0.0551▲</b> | 0.0237▼        | <b>0.0282▲</b> |
| URItem-OM                  | <b>0.0498▲</b> | <b>0.0536▲</b> | 0.0234▼        | <b>0.0280▲</b> |
| URUser-OM                  | <b>0.0518▲</b> | <b>0.0551▲</b> | 0.0234▼        | <b>0.0279▲</b> |
| URIItemUser-OM             | <b>0.0505▲</b> | <b>0.0542▲</b> | 0.0235▼        | <b>0.0280▲</b> |
| Entropy-FW                 | 0.0344▼        | 0.0410▼        | <b>0.0241△</b> | <b>0.0275▲</b> |
| UserSimple-FW              | <b>0.0435△</b> | <b>0.0503△</b> | <b>0.0244▲</b> | <b>0.0276▲</b> |
| UserItem-FW                | <b>0.0435▼</b> | <b>0.0501△</b> | <b>0.0245▲</b> | <b>0.0275▲</b> |
| RatItem-FW                 | <b>0.0436▼</b> | <b>0.0504△</b> | <b>0.0244▲</b> | <b>0.0275▲</b> |
| RatUser-FW                 | <b>0.0440▼</b> | <b>0.0509△</b> | <b>0.0245▲</b> | <b>0.0276▲</b> |
| URItem-FW                  | <b>0.0429▼</b> | <b>0.0494▲</b> | <b>0.0244▲</b> | <b>0.0273△</b> |
| URUser-FW                  | <b>0.0438▼</b> | <b>0.0506△</b> | <b>0.0245▲</b> | <b>0.0274▲</b> |
| URIItemUser-FW             | <b>0.0432▼</b> | <b>0.0498▲</b> | <b>0.0245▲</b> | <b>0.0274▲</b> |

**Table 7.9. Dynamic ensemble performance values (P@10) using uuU1R methodology with item predictors (MovieLens dataset).**

In the other combination (HRI3) the best strategy is FW, the same as with the other evaluation methodologies.

### 7.3.2 Dynamic recommender ensembles on log data

In this section we present experiments in which log-based predictors are used to dynamically weight an ensemble's recommenders. As with rating-based information, in this case we first have to select suitable recommenders to combine according to the requirements established in our framework. Hence, we choose the combinations HL1, HL2 and HL3 presented in Table 7.10, where, as before, the performance predictors weight the recommender denoted as R1.

The Last.fm dataset contains timestamped log-based information. As noted in Chapter 4, for efficiency reasons, we only use the 1R methodology in this dataset. Table 7.11 shows the results obtained with a temporal split of the data, and Table 7.12 shows the results obtained with a random split (five-fold) of the data.

|     | R1   | R2      |
|-----|------|---------|
| HL1 | kNN  | CB      |
| HL2 | kNN  | ItemPop |
| HL3 | pLSA | kNN     |

**Table 7.10. Selected recommenders for building dynamic ensembles using performance predictors that exploit log-based information (Last.fm dataset).**

|                                  | HL1             | HL2                    | HL3             |
|----------------------------------|-----------------|------------------------|-----------------|
| R1 ( $\lambda=1.0$ )             | 0.0603          | 0.0603                 | <u>0.0926</u>   |
| R2 ( $\lambda=0.0$ )             | <u>0.0916</u>   | 0.0797                 | 0.0603          |
| Baseline ( $\lambda=0.5$ )       | 0.0852          | 0.0755                 | 0.0820          |
| Best static<br>(best $\lambda$ ) | 0.0914<br>(0.2) | <u>0.0812</u><br>(0.1) | 0.0925<br>(0.9) |
| Perfect correlation              | 0.0890          | 0.0783                 | 0.0863          |
| PC-OM                            | 0.0869          | 0.0771                 | 0.0851          |
| PC-FW                            | 0.0849          | 0.0751                 | 0.0826          |
| ItemSimple-OM                    | <b>0.0904▼</b>  | <b>0.0804▼</b>         | <b>0.0901▼</b>  |
| Autocorrelation-OM               | 0.0815▼         | 0.0722▼                | 0.0781▼         |
| TimeSimple-OM                    | <b>0.0905▼</b>  | <b>0.0789▼</b>         | <b>0.0898▼</b>  |
| ItemTime-OM                      | <b>0.0906▼</b>  | <b>0.0804▼</b>         | <b>0.0902▼</b>  |
| ItemPriorTime-OM                 | <b>0.0885▼</b>  | <b>0.0778▼</b>         | <b>0.0863▼</b>  |
| ItemSimple-FW                    | <b>0.0903▼</b>  | <b>0.0802▼</b>         | <b>0.0891▼</b>  |
| Autocorrelation-FW               | 0.0842▼         | 0.0746▼                | 0.0809▼         |
| TimeSimple-FW                    | <b>0.0901▼</b>  | <b>0.0785▼</b>         | <b>0.0884▼</b>  |
| ItemTime-FW                      | <b>0.0904▼</b>  | <b>0.0800▼</b>         | <b>0.0891▼</b>  |
| ItemPriorTime-FW                 | <b>0.0883▼</b>  | <b>0.0775▼</b>         | <b>0.0855▼</b>  |

**Table 7.11. Dynamic ensemble performance values (P@10) using the 1R methodology with the log-based user predictors (Last.fm, temporal split).**

|                                  | HL1             | HL2                    | HL3                    |
|----------------------------------|-----------------|------------------------|------------------------|
| R1 ( $\lambda=1.0$ )             | 0.0204          | 0.0204                 | 0.0836                 |
| R2 ( $\lambda=0.0$ )             | <u>0.0828</u>   | <u>0.0767</u>          | 0.0204                 |
| Baseline ( $\lambda=0.5$ )       | 0.0764          | 0.0643                 | 0.0704                 |
| Best static<br>(best $\lambda$ ) | 0.0818<br>(0.2) | <u>0.0767</u><br>(0.1) | <u>0.0837</u><br>(0.9) |
| Perfect correlation              | 0.0818          | 0.0760                 | 0.0829                 |
| PC-OM                            | 0.0816          | 0.0755                 | 0.0823                 |
| PC-FW                            | 0.0815          | 0.0745                 | 0.0811                 |
| ItemSimple-OM                    | <b>0.0799▼</b>  | <b>0.0730▼</b>         | <b>0.0771▼</b>         |
| Autocorrelation-OM               | 0.0717▼         | 0.0596▼                | 0.0686▼                |
| TimeSimple-OM                    | <b>0.0814▼</b>  | <b>0.0762▼</b>         | 0.0518▼                |
| ItemTime-OM                      | <b>0.0806▼</b>  | <b>0.0734▼</b>         | <b>0.0761▼</b>         |
| ItemPriorTime-OM                 | <b>0.0770▼</b>  | <b>0.0658▼</b>         | <b>0.0743▼</b>         |
| ItemSimple-FW                    | <b>0.0804▼</b>  | <b>0.0726▼</b>         | <b>0.0739▼</b>         |
| Autocorrelation-FW               | 0.0756▼         | 0.0631▼                | 0.0697▼                |
| TimeSimple-FW                    | <b>0.0814▼</b>  | <b>0.0753▼</b>         | 0.0579▼                |
| ItemTime-FW                      | <b>0.0808▼</b>  | <b>0.0728▼</b>         | <b>0.0732▼</b>         |
| ItemPriorTime-FW                 | <b>0.0783▼</b>  | <b>0.0671▼</b>         | <b>0.0719▼</b>         |

**Table 7.12. Dynamic ensemble performance values (P@10) using the 1R methodology with log-based user predictors (Last.fm, five-fold random split).**

We can see that the results of both tables are analogous. **The dynamic ensembles weighted by the log-based performance predictors outperform the baseline static ensemble in all cases, except with the Autocorrelation predictor.** This result is consistent with the correlations presented in Table 6.14 and Table 6.15, where autocorrelation obtained the lowest (absolute) correlation value for the kNN recommender on both versions of the dataset. Regarding the pLSA recommender (in the combination HL3), the Autocorrelation and TimeSimple predictors obtain com-

|     | R1         | R2   |
|-----|------------|------|
| HS1 | Personal   | pLSA |
| HS2 | Personal   | kNN  |
| HS3 | PureSocial | pLSA |
| HS4 | PureSocial | kNN  |

**Table 7.13. Selected recommenders for building dynamic ensembles using social-based user predictors (CAMRa dataset).**

parable correlations with the combined recommenders, yet the performance of the corresponding dynamic ensembles is very different, thus suggesting that, although we have found a dependence between the predictors' power in terms of correlation, and their effectiveness in weighting hybrids, this is not a strict necessary condition to obtain improvements over the static ensembles.

The best performance values were achieved either by single recommenders or by the best static ensembles. When the best results are obtained by single recommenders emphasises the fact that no hybridisation is required for that combination (like in HL1 and HL3 for the temporal split, and HL1 and HL2 for the random split). In the other case, when the best results are achieved by the best static ensembles, it may restrict the usefulness of our approach, although our proposed dynamic ensembles significantly outperform the baseline static ensembles for some predictors such as TimeSimple and ItemSimple. We have to recall that the best static ensembles are in fact optimised using the test set, which is clearly not a fair comparison. The results of the perfect correlation ensembles in the random split are always better than those obtained by the performance predictors, confirming that predictors with stronger correlations should obtain better performance results when used for dynamic ensembles.

### 7.3.3 Dynamic recommender ensembles on social data

In the third experiment we exploit the social information available in the CAMRa dataset to combine collaborative and social filtering recommenders using social-based performance predictors. Table 7.13 shows the recommender combinations selected based on the correlations obtained in Section 6.5.4. Here, we present 4 ensembles where the two social filtering recommenders, Personal and PureSocial, are combined with two collaborative filtering recommenders, pLSA and kNN. We saw in Section 6.5.4 that most of the social-based predictors obtained higher correlations with the social filtering recommenders, and lower or negligible correlations with the collaborative filtering recommenders, at least for the social version of the dataset (Table 6.16). The situation for the collaborative-social version was not so clear, but for the sake of coherence, we use the same set of ensembles in both versions of the dataset.

|                                  | HS1                 | HS2                                     | HS3                                 | HS4                                 |
|----------------------------------|---------------------|---|-------------------------------------|-------------------------------------|
| R1 ( $\lambda=1.0$ )             | 0.1732              | 0.1732                                  | 0.1760                              | 0.1760                              |
| R2 ( $\lambda=0.0$ )             | 0.1110              | 0.0473                                  | 0.1110                              | 0.0473                              |
| Baseline ( $\lambda=0.5$ )       | 0.1813              | 0.1821                                  | 0.2006                              | 0.1929                              |
| Best static<br>(best $\lambda$ ) | 0.1842<br>(0.7)     | 0.1899<br>(0.8)                         | 0.2012<br>(0.4)                     | 0.1952<br>(0.6)                     |
| Perfect correlation              | <u>0.2018</u>       | <u>0.1929</u>                           | <u>0.2089</u>                       | <u>0.1979</u>                       |
| PC-OM                            | 0.1872              | 0.1875                                  | 0.2048                              | 0.1946                              |
| PC-FW                            | 0.1863              | 0.1869                                  | 0.2042                              | 0.1994                              |
| AvgNeighDeg-OM                   | 0.1795 $\downarrow$ | <b>0.1896<math>\triangle</math></b>     | 0.1973 $\downarrow$                 | 0.1804 $\downarrow$                 |
| BetCentrality-OM                 | 0.1744 $\downarrow$ | 0.1804 $\downarrow$                     | 0.1833 $\downarrow$                 | 0.1777 $\downarrow$                 |
| ClustCoeff-OM                    | 0.1786 $\downarrow$ | 0.1786 $\downarrow$                     | 0.1836 $\downarrow$                 | 0.1753 $\downarrow$                 |
| Degree-OM                        | 0.1738 $\downarrow$ | <b>0.1839<math>\triangledown</math></b> | 0.1976 $\downarrow$                 | 0.1765 $\downarrow$                 |
| EgoCompSize-OM                   | 0.1756 $\downarrow$ | <b>0.1833<math>\triangledown</math></b> | 0.1967 $\downarrow$                 | 0.1827 $\downarrow$                 |
| HITS-OM                          | 0.1774 $\downarrow$ | <b>0.1911<math>\triangle</math></b>     | 0.1813 $\downarrow$                 | 0.1798 $\downarrow$                 |
| PageRank-OM                      | 0.1762 $\downarrow$ | <b>0.1842<math>\triangledown</math></b> | 0.1917 $\downarrow$                 | 0.1801 $\downarrow$                 |
| TwoHopNeigh-OM                   | 0.1756 $\downarrow$ | <b>0.1851<math>\triangle</math></b>     | 0.1964 $\downarrow$                 | 0.1777 $\downarrow$                 |
| AvgNeighDeg-FW                   | 0.1807 $\downarrow$ | <b>0.1896<math>\triangle</math></b>     | 0.2003 $\downarrow$                 | 0.1914 $\downarrow$                 |
| BetCentrality-FW                 | 0.1801 $\downarrow$ | <b>0.1872<math>\triangle</math></b>     | <b>0.2024<math>\triangle</math></b> | <b>0.1929<math>\triangle</math></b> |
| ClustCoeff-FW                    | 0.1804 $\downarrow$ | <b>0.1875<math>\triangle</math></b>     | 0.2003 $\downarrow$                 | 0.1890 $\downarrow$                 |
| Degree-FW                        | 0.1798 $\downarrow$ | <b>0.1887<math>\triangle</math></b>     | 0.2000 $\downarrow$                 | <b>0.1929<math>\triangle</math></b> |
| EgoCompSize-FW                   | 0.1789 $\downarrow$ | <b>0.1896<math>\triangle</math></b>     | <b>0.2009<math>\triangle</math></b> | <b>0.1938<math>\triangle</math></b> |
| HITS-FW                          | 0.1801 $\downarrow$ | <b>0.1902<math>\triangle</math></b>     | 0.1997 $\downarrow$                 | 0.1926 $\downarrow$                 |
| PageRank-FW                      | 0.1810 $\downarrow$ | <b>0.1875<math>\triangle</math></b>     | 0.2003 $\downarrow$                 | 0.1923 $\downarrow$                 |
| TwoHopNeigh-FW                   | 0.1801 $\downarrow$ | <b>0.1905<math>\triangle</math></b>     | 0.2000 $\downarrow$                 | 0.1926 $\downarrow$                 |

**Table 7.14. Dynamic ensemble performance values (P@10) using the AR methodology with social-based user predictors (CAMRa, social dataset).**

As we mentioned in Section 6.5.4, due to the lack of coverage of the social filtering recommenders, the only methodology that provides sensible results is the AR methodology. In this section we present the results obtained using this methodology on the two available versions of the CAMRa dataset: social and collaborative-social.

Table 7.14 shows the results obtained on the social version of the CAMRa dataset. We see that only for one out of the four recommender combinations, the dynamic ensembles consistently outperform the baseline static ensemble. However, it is interesting to note that the best value is always achieved by the perfect correlation ensemble, which means that further improvements could be possible if we were able to find predictors with stronger correlations.

In the collaborative-social version of the dataset (Table 7.15) the results are similar, except that now for HS2, the best result is obtained by the best static ensemble. Moreover, a larger number of dynamic ensembles outperform the baseline static ensemble HS3, whereas at least one dynamic ensemble outperforms the baseline HS1, which is a better result than the one shown in the previous Table 7.14. We hypothesise this is because on this version of the dataset the individual recommenders display a more similar performance to each other (compare the differences between R1 and R2 in Table 7.14 and Table 7.15).

Furthermore, some of the correlations obtained for the CAMRa collaborative dataset are more discriminative between the combined recommenders, in the sense that, for instance, the correlations between the two-hop neighbourhood predictor and the Personal recommender were -0.123 and -0.121 in the social and collaborative-social datasets, respectively. However, the correlations between the two-hop neighbourhood predictor and kNN were 0.004 and 0.130, that is, in the second dataset the relative distance in correlation between these two recommenders is larger, according to the correlation with respect to the predictor. This change in the correlations may explain the fact that in Table 7.15 some of the dynamic ensembles outperform the perfect correlation ensemble, which does not take the relative correlation into account with respect to each individual recommender, as noted in 7.3.1.

In general, the HITS predictor obtains the best results among the dynamic ensembles for some of the tested combinations. Other predictors such as the betweenness centrality and the ego components size produce more competitive ensembles in the social version of the dataset, whereas the degree and the average neighbour degree predictors provide better results for more than one combination in the CAMRa collaborative dataset.

|                                  | HS1                                 | HS2   | HS3                                 | HS4                         |
|----------------------------------|-------------------------------------|---|-------------------------------------|-----------------------------|
| R1 ( $\lambda=1.0$ )             | 0.1066                              | 0.1066  | 0.1072                              | 0.1072                      |
| R2 ( $\lambda=0.0$ )             | 0.1007                              | 0.0226  | 0.1007                              | 0.0226                      |
| Baseline ( $\lambda=0.5$ )       | 0.1509                              | 0.1142  | 0.1599                              | 0.1219                      |
| Best static<br>(best $\lambda$ ) | 0.1524<br>(0.4)                     | <u>0.1200</u><br>(0.7)                          | 0.1632<br>(0.3)                     | 0.1219<br>(0.5)             |
| Perfect correlation              | <u>0.1608</u>                       | 0.1188  | <u>0.1640</u>                       | <u>0.1237</u>               |
| PC-OM                            | 0.1202                              | 0.1164  | 0.1254                              | 0.1199                      |
| PC-FW                            | 0.1189                              | 0.1143  | 0.1263                              | 0.1219                      |
| AvgNeighDeg-OM                   | 0.1489 $\triangledown$              | <b>0.1195<math>\triangle</math></b>             | 0.1599 $\triangledown$              | 0.1131 $\triangledown$      |
| BetCentrality-OM                 | 0.1443 $\blacktriangledown$         | 0.1132 $\triangledown$                          | 0.1487 $\blacktriangledown$         | 0.1114 $\blacktriangledown$ |
| ClustCoeff-OM                    | 0.1465 $\triangledown$              | 0.1123 $\triangledown$                          | 0.1483 $\blacktriangledown$         | 0.1108 $\blacktriangledown$ |
| Degree-OM                        | 0.1472 $\triangledown$              | <b>0.1154<math>\triangle</math></b>             | <b>0.1614<math>\triangle</math></b> | 0.1107 $\blacktriangledown$ |
| EgoCompSize-OM                   | 0.1461 $\triangledown$              | <b>0.1158<math>\triangle</math></b>             | 0.1596 $\triangledown$              | 0.1140 $\blacktriangledown$ |
| HITS-OM                          | 0.1485 $\triangledown$              | <u><b>0.1200<math>\blacktriangle</math></b></u> | 0.1467 $\blacktriangledown$         | 0.1134 $\blacktriangledown$ |
| PageRank-OM                      | 0.1471 $\blacktriangledown$         | <b>0.1167<math>\triangle</math></b>             | 0.1579 $\triangledown$              | 0.1123 $\blacktriangledown$ |
| TwoHopNeigh-OM                   | 0.1478 $\triangledown$              | <b>0.1171<math>\triangle</math></b>             | 0.1585 $\blacktriangledown$         | 0.1118 $\blacktriangledown$ |
| AvgNeighDeg-FW                   | <b>0.1518<math>\triangle</math></b> | <b>0.1191<math>\triangle</math></b>             | <b>0.1623<math>\triangle</math></b> | 0.1204 $\blacktriangledown$ |
| BetCentrality-FW                 | 0.1491 $\triangledown$              | <b>0.1180<math>\triangle</math></b>             | 0.1577 $\triangledown$              | 0.1213 $\blacktriangledown$ |
| ClustCoeff-FW                    | 0.1500 $\triangledown$              | <b>0.1182<math>\triangle</math></b>             | 0.1566 $\blacktriangledown$         | 0.1189 $\blacktriangledown$ |
| Degree-FW                        | 0.1489 $\triangledown$              | <b>0.1191<math>\triangle</math></b>             | <b>0.1627<math>\triangle</math></b> | 0.1208 $\blacktriangledown$ |
| EgoCompSize-FW                   | 0.1489 $\triangledown$              | <b>0.1193<math>\triangle</math></b>             | <b>0.1618<math>\triangle</math></b> | 0.1210 $\blacktriangledown$ |
| HITS-FW                          | 0.1482 $\blacktriangledown$         | <b>0.1195<math>\triangle</math></b>             | 0.1564 $\blacktriangledown$         | 0.1202 $\blacktriangledown$ |
| PageRank-FW                      | 0.1491 $\triangledown$              | <b>0.1186<math>\triangle</math></b>             | <b>0.1610<math>\triangle</math></b> | 0.1211 $\blacktriangledown$ |
| TwoHopNeigh-FW                   | 0.1500 $\triangledown$              | <b>0.1195<math>\triangle</math></b>             | <b>0.1619<math>\triangle</math></b> | 0.1211 $\blacktriangledown$ |

**Table 7.15. Dynamic ensemble performance values (P@10) using the AR methodology with social-based user predictors (CAMRa, collaborative dataset).**

### 7.3.4 Discussion

The analysis of the results presented in this chapter shows that ensembles can indeed benefit from a dynamic weighting of their recommenders. In particular, we have seen that when these weights come from performance predictors, which previously had shown significant correlation with the performance of individual recommenders, the resulting dynamic ensemble tends to outperform static combinations of the recommenders. In this context, in order to obtain successful hybridisations, we have to take several variables into account, which correspond to three stages proposed in our framework: the correlation between the predictor and the combined recommenders, the relative performance of such recommenders, the strategy to normalise the predictor's values, and the weight distribution among recommenders.

The relative performance of the recommenders has proven to be decisive, since in some cases, hybridisation does not make sense to begin with, when the difference in performance between the recommenders is significant and systematic, and thus, dynamic ensembles cannot obtain the best performance result, although they may outperform static ensembles. Performance prediction normalisation and weight distribution, on the other hand, do make a difference in the results. Although no explicit results are presented in this work regarding different normalisation approaches, previously conducted experiments showed us that score normalisation produce worse results than rank normalisation. Finally, the weight distribution strategy is not as critical as other stages of our framework, but helps to obtain much better results, specifically, when the one minus strategy (OM) is used.

The obtained results have also shown that more complex formalisations and probability models do not necessarily lead to better results, with respect to the adaptation and definition of the user and item clarity performance predictors. In this adaptation, various configurations were available, and we experimented with further extensions of different language models for the same clarity model, using rating and log-based information. Additionally, several graph-based metrics were tested, where the concept of the user's strength in a social network is modelled in different ways.

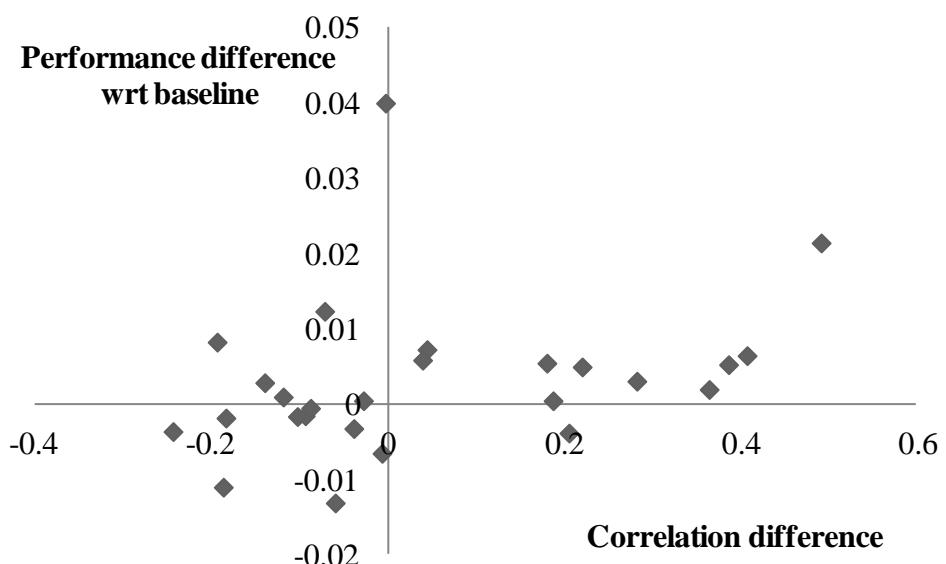
We find that different formulations for the user-based performance clarity predictor consistently obtain the best results in different situations for rating-based preference information. We also experimented with item-based predictors, and found that the UserItem, URItem, and RatUser predictors were noticeably better than the rest of the formulations. When log-based information is exploited, the ItemTime and TimeSimple predictors obtained better results than other predictors not based on the clarity concept, such as the Autocorrelation function. Moreover, regarding the social-based ensembles, the HITS, two-hop neighbourhood, and average neighbour degree approaches clearly outperform the ensemble weighted by the rest of the predictors and, in most of the cases, also outperform the baseline static ensemble.

These results are, in general, consistent with the correlation values between the predictors' output values and the recommenders' performance values. Figure 7.2 shows a summary of the results presented in this and previous chapters, where the difference in correlation is plotted against the gain (or loss) in performance with respect to the baseline. For this figure, the best and worst dynamic ensembles were selected from Table 7.2, Table 7.11 and Table 7.15. In the figure we may observe the trend that the larger the difference in correlation, the better the improvement over the baseline, which is in concordance with the requirement that both correlations should not be very similar. These results provide some insights in order to understand which features may help configure well performing dynamic recommender ensembles, where performance predictors have emerged as a clear useful characteristic.

## 7.4 Conclusions

In this chapter we have explored how the performance of a recommender ensemble can be improved by dynamically assigning the weights of its recommenders, by analysing the performance correlation between the values of a performance predictor and the performance of an individual recommender. In this way, we have proposed a dynamic hybrid framework that let decide when and how dynamic hybridisation should be done.

Drawing from the performance predictors proposed in the previous chapter, we have conducted several experiments in order to assess whether recommender en-



**Figure 7.2.** For each best and worst dynamic ensemble in Table 7.2, Table 7.11 and Table 7.15, this graph plots the difference in correlation between each predictor and a recommender against the difference in performance between the ensemble and the baseline.

sembles can benefit from dynamic weights according to such predictors. The results obtained in our experiments indicate that a strong correlation with performance tends to correspond with enhancements in ensembles by using the predictor for weight adjustment. The dynamic ensembles usually outperformed the baseline static ensemble for different recommender combinations, supporting their effectiveness in different situations.

In future work we aim to evaluate our framework with more than two recommenders in an ensemble, and more than one performance predictor, eventually, one for each recommender. We also plan to test different normalisation strategies of the predictor's values, where several assumptions about the ideal weight distribution can be verified, such as whether the user's rating distribution or the recommender's output are beneficial for the final performance of the ensemble. Moreover, Machine Learning approaches could also be used to learn the best weights in a user (or item) basis. Despite being more time consuming, these techniques may also achieve good results in terms of performance of the dynamic ensemble, although they are usually more prone to overfit the learned weights.

# Chapter 8

## Neighbour selection and weighting in user-based collaborative filtering

User-based recommender systems suggest interesting items to a user relying on similar-minded people called neighbours. The selection and weighting of the input from these neighbours characterise different variants of the approach. Thus, for instance, while standard user-based collaborative filtering strategies select neighbours based on user similarities, trust-aware recommendation algorithms rely on other aspects indicative of user trustworthiness and reliability.

In this chapter we restate the user-based recommendation problem, generalising it in terms of performance prediction techniques. We investigate how to adopt this generalisation to define a unified framework where we conduct an objective analysis of the effectiveness (predictive power) of neighbour scoring functions. We evaluate our approach with several state-of-the-art and novel neighbour scoring functions on two publicly available datasets. The notion of performance takes here a different nuance from previous chapters. More precisely, we consider the notion of neighbour performance, for which we propose several measures and new predictors. In an empirical comparison involving four neighbour quality metrics and thirteen performance predictors, we find a strong predictive power for some of the predictors with respect to certain metrics. This result is then validated by checking the final performance of recommendation strategies where predictors are used for selecting and/or weighting user neighbours. As a result, we are able to anticipate which predictors will perform better in neighbour scoring powered versions of a user-based collaborative filtering algorithm.

In Sections 8.1 and 8.2 we present a unified formulation and the proposed framework for neighbour selection and weighting in user-based recommendation, and in Section 8.3 we describe how the different neighbour scoring functions proposed in the literature fit into the framework. Finally, in Section 8.4 we present an experimental evaluation of the framework, and in Section 8.5 we provide conclusions.

## 8.1 Problem statement

We focus on user-based collaborative filtering algorithms, one type of memory-based approaches that explicitly seek people – commonly called **neighbours** – having preferences (and/or other characteristics of interest) in common with the target user, and use such preferences to predict item ratings for the user. User-based algorithms are built on the principle that a particular user's rating records are not equally useful to all other users as input to provide them with item suggestions (Herlocker et al., 2002). Therefore, as stated in Chapter 2, central aspects to these algorithms are a) how to identify which neighbours form the best basis to generate item recommendations for the target user, and b) how to properly make use of the information provided by them. Once the target user's neighbours are selected, the more similar a neighbour is to the user, the more her preferences are taken into account as input to produce recommendations.

A common user-based recommendation approach consists of predicting the relevance of an item for the target user by a linear combination of her neighbours' ratings, which are weighted by the similarity between the target user and her neighbours, as presented in Equation (2.3). For the sake of clarity, and since we shall later elaborate from it, we reproduce here the above equation:

$$\tilde{r}(u, i) = \bar{r}(u) + C \sum_{v \in N_k(u, i)} sim(u, v)(r(v, i) - \bar{r}(v)) \quad (8.1)$$

User similarity has been the central criterion for neighbour selection in most of the user-based collaborative filtering approaches (Desrosiers and Karypis, 2011). Nonetheless, recently it has been suggested that additional factors could have a valuable role to play on this point. For instance, two users with a high similarity value may no longer be reliable predictors for each other at some point because of a divergence of tastes over time (O'Donovan and Smyth, 2005). Thus, in the context of user-based collaborative filtering, more complex methods have been proposed in order to effectively select and weight useful neighbours (O'Donovan and Smyth, 2005; Desrosiers and Karypis, 2011). In this context a particularly relevant dimension relates the above additional factors with the general concept of trust (trustworthiness, reputation) on a user's contribution to the computation of recommendations. Hence, a number of trust-aware recommender systems have been proposed in the last decade (Hwang and Chen, 2007; O'Donovan and Smyth, 2005; Golbeck, 2009).

Most of these systems focus on the improvement of accuracy metrics, such as the Mean Average Error, by defining different heuristic trust functions, which, in most cases, are applied either as additional weighting factors in the neighbourhood-based formulation, or as a component of the neighbour selection criteria. The way trust is measured is considerably diverse in the literature. In fact, the notion of trust

has embraced a wide scope of neighbour aspects, spanning from personal trust on the neighbour's faithfulness, to trust on her competence, confidence in the correctness of the input data, or the effectiveness of the recommendation resulting from the neighbour's data. More specifically, in trust-aware recommender systems, a trust model is defined and, typically, introduced into the Resnick's equation (Equation (8.1)) either as an additional weight or as a filter for the potential user's neighbours. Moreover, depending on the nature of their input, different types of trust-aware recommendation approaches can be distinguished: rating-based approaches, and social-based approaches (using a trust network).

One of the first works that proposed *rating-based trust metrics* between users is (O'Donovan and Smyth, 2005). In that work O'Donovan and Smyth propose to modify how the "recommendation partners" (neighbours) are weighted and selected in the user-based collaborative filtering formula. They argue that the trustworthiness of a particular neighbour should be taken into account in the computed recommendation score by looking at how reliable her past recommendations were. Trust values are computed by measuring the amount of correct recommendations in which a user has participated as a neighbour, and then they are used for weighting the influence (along with computing the similarity), and selecting the target user's neighbours. Weng et al. (2006) propose an asymmetric trust metric based on the expectation of other users' competence in providing recommendations to reduce the uncertainty in predicting new ratings. The metric is used in the standard collaborative filtering formula instead of the similarity value. Two additional metrics are defined in (Kwon et al., 2009) based on the similarity between the ratings of a neighbour and the ratings from the community. Finally, Hwang and Chen (2007) define two trust metrics (local and global) by averaging the prediction error of co-rated items between a user and a potential neighbour.

*Social-based trust metrics* make use of explicit trust networks of users, built upon friendship relations (Massa and Avesani, 2004; Massa and Bhattacharjee, 2004) and explicit trust scores between individuals in a system (Ma et al., 2009; Walter et al., 2009). These metrics and, to some extent, their inherent meanings, are different with respect to rating-based metrics. Nonetheless, Ziegler and Lausen (2004) conduct a thorough analysis that shows empirical correlations between trust and user similarities, suggesting that users tend to create social connections with people who have similar preferences. Once such a correlation is proved, techniques based on social-based trust can be applicable. Golbeck and Hendler (2006) propose a metric called TidalTrust to infer trust relationships by using recursive search. Inferred trust values are used for every user who has rated a particular item in order to select only those users with high trust values. Then, a weighted average between past ratings and inferred trust values provides the predicted ratings. Massa and Avesani (2007b) ex-

periment with local (MoleTrust) and global (PageRank) trust metrics, showing that trust-based recommenders are very valuable for cold start users.

The research presented here seeks to provide an algorithmic generalisation for a significant variety of notions, computational definitions, and roles of trust in neighbour selection. Specifically, we aim to provide a theoretical framework for neighbour selection and weighting in which trust metrics can be defined and evaluated in terms of improvements on a final recommender's performance. We cast the rating prediction task – typically based, as described above, on the aggregation of neighbour preferences – into a framework for dynamic combination of inputs, from a performance prediction perspective, borrowing from the methodology for this area in the Information Retrieval field. The application of this perspective is not trivial, and requires a definition of what the performance of a neighbour means in this context. Hence, restated the problem in these terms, we propose to adapt and exploit techniques and methodologies developed in Information Retrieval for predicting query performance; in our case the target user's neighbours are equivalent to the queries, and our goal is to predict which of these neighbours will perform better for the target user.

Furthermore, since our framework provides an objective measure of the neighbour scoring function efficiency, we would be able to obtain a better understanding of the whole recommendation process. For instance, if the results obtained when a particular function is introduced in a recommender are not consistent with the (already observed) objective performance measures, it would mean that the chosen strategy is not the most appropriate, suggesting to experiment with further strategies, providing such a function has already shown some predictive power.

Therefore, the main contribution of our framework is that it provides a formal setting for the evaluation of neighbour selection and weighting functions, while, at the same time, enables to discriminate whether recommendation performance improvements are achieved by the neighbour scoring functions, or by the way these functions are used in the recommendation computation. Besides, our framework provides an unification of state-of-the-art trust-based recommendation approaches, where trust metrics are casted as neighbour performance predictors. As a result, in this chapter, we shall propose four neighbour quality metrics and thirteen performance predictors, defined upon a specific neighbour (user-based), a neighbour and the current user (user-user), or a neighbour and the current item (user-item). We shall generalise the different strategies proposed in the literature to introduce trust into collaborative filtering. Moreover, thanks to the proposed formulation, we will define and evaluate new strategies.

## 8.2 A performance prediction framework for neighbour scoring

### 8.2.1 Unifying neighbour selection and weighting in user-based Recommender Systems

From the observation that most of the methods for neighbour selection and weighting are elaborated upon the standard Resnick's scheme (Equation (8.1)), we propose a unified formulation as follows. Let us suppose, for the sake of generality, that we have a neighbour scoring function  $s(u, v, i)$  that may depend on the target user  $u$ , a neighbour  $v$ , and a target item  $i$ . This function outputs a higher value whenever the user, neighbour, item, or a combination of them, is more trustworthy (in the case of trust models), or is expected to perform better as a neighbour according to the information available in the system, such as other ratings and external information, like a social network. Using this function we generalise Equation (8.1) to:

$$\tilde{r}(u, i) = \bar{r}(u) + C \sum_{v \in f^{neigh}(u, i; k; s)} f^{agg}(s(u, v, i), sim(u, v))(r(v, i) - \bar{r}(v)) \quad (8.2)$$

where the function  $f^{neigh}$  denotes the selection of the set of neighbours, and  $f^{agg}$  is an aggregation function combining the output of  $s$  and the user similarity into a single weight value. In this way, we integrate the neighbour scoring function  $s$  into the Resnick's formula in order to: a) select the neighbours to be considered, instead of or in addition to the most similar users (via function  $f^{neigh}$ ), and b) provide a general weighting scheme by introducing an aggregation function  $f^{agg}$  between the actual neighbour score and the similarity between the target user and her neighbours. Note that it is not required that  $s$  is bounded, since a constant  $C$  would normalise the output rating value. The function  $s$  is thus a core component in the generalisation of the user-based collaborative filtering techniques. It may embody similarity in itself (in such case  $f^{agg}$  may just return its first input argument), but  $sim$  and  $f^{agg}$  are left to simplify the connection with the original similarity-only formulation, and to suit particular cases where  $s$  applies other principles distinct to similarity.

The aggregation function  $f^{agg}$  can take different definitions, some examples of which can be found in the literature. For instance, O'Donovan and Smyth (2005) initially propose to use the arithmetic mean of the neighbour score ( $x$ ) and the similarity ( $y$ ; henceforth denoted as  $f_1^{agg}$ ), and end up using the harmonic mean ( $f_2^{agg}$ ) because of its better robustness to large differences in the inputs. In (Bellogín and Castells, 2010), on the other hand, we use the product function ( $f_3^{agg}$ ). Moreover, Hwang and Chen (2007) propose to directly use the neighbour score as the weight

given to neighbours, that is, they use the projection function  $f_4^{agg}(x, y) = x$ . Obviously, the original Resnick's formulation can be expressed as the symmetric projection function  $f_0^{agg}(x, y) = y$ .

The neighbourhood selection embodied in function  $f^{neigh}$  also generalises Resnick's approach – the latter corresponds to the particular case  $f_0^{neigh}(u, i; k; s) = N_k(u, i)$ , where the neighbour scoring function is ignored, and only similarity is used. The general form admits different instantiations. In (Golbeck and Handler, 2006) only the users with the highest trust values are selected as neighbours. In (O'Donovan and Smyth, 2005), on the other hand, those users whose trust values exceed a certain threshold are taken into consideration. This threshold is empirically defined as the mean across all the obtained values for each pair of users. The latter strategy can be formulated as follows:

$$f_1^{neigh}(u, i; k; s) = \{v \in N_k(u, i) : s(u, v, i) > \tau\}; \tau = \frac{1}{|\{(u, v, i)\}|} \sum_{(u, v, i)} s(u, v, i)$$

There are, nonetheless, some considerations to take into account when using specific combinations of neighbour weighting and neighbour selection functions. First, if  $f_4^{agg}$  is used together with  $f_0^{neigh}$  – only considering the most similar users in the neighbourhood –, then less reliable users (with low  $f_4^{agg}$ ) who are very similar to the current user would be penalised, and more reliable neighbours but less similar to the current user are ignored, since they do not belong to the neighbourhood. Second, when using  $f_0^{agg}$  together with  $f_1^{neigh}$ , neighbours are weighted by their similarities with the target user. These similarities, however, could be very low, and thus, non-similar but reliable neighbours would be penalised. Finally, if  $f_4^{agg}$  is used with  $f_1^{neigh}$ , the similarity weight will not be considered at any point in the recommendation process.

Some of these configurations may deserve further investigation, and are considered in Section 8.4, along with other combinations not listed here.

## 8.2.2 Neighbour selection and weighting as a performance prediction problem

Neighbour scoring and selection can be seen as a task of predicting the effectiveness of neighbours as input for collaborative recommendations. In this section we elaborate and adapt the performance prediction framework presented in Chapter 5 to the problem of neighbour selection and weighting.

The same as performance prediction in Information Retrieval, which has been used to optimise rank aggregation (Yom-Tov et al., 2005a), in our proposed framework each user's neighbour can be considered as a retrieval subsystem (or criterion)

whose output is combined to form a final system's output (the recommendations) to the user.

For user-based collaborative filtering algorithms, the estimation  $\tilde{r}(u, i)$  of the preference of the target user  $u$  for a particular item  $i$  can be formulated as an aggregation function of the ratings of some other users  $\hat{V}$ :

$$\tilde{r}(u, i) \propto \text{aggr}_{v \in \hat{V}}(\text{sim}(u, v); r(v, i); \bar{r}(u); \bar{r}(v)) \quad (8.3)$$

where  $\hat{V}$  denotes the selected neighbours for a particular user  $u$  according to function  $f^{neigh}$  (see Equation (8.2)). As observed in (Adomavicius and Tuzhilin, 2005), different aggregation functions can be defined, but the most typical one is the weighted average function presented in the previous section.

In the previous function the term  $\tilde{r}(u, i)$  can be seen as a retrieval function that aggregates the outputs of several utility subfunctions  $r(v, i) - \bar{r}(v)$ , each corresponding to a recommendation obtained from a neighbour of the target user. The combination of utility values is defined as a linear combination (translated by  $\bar{r}(u)$ ) of the neighbours' ratings, weighted by their similarity  $\text{sim}(u, v)$  with the target user. Hence, the computation of utility values in user-based filtering is equivalent to a typical rank aggregation model of Information Retrieval, where the aggregated results may be enhanced by predicting the performance of the combined recommendation outputs. In fact, the similarity value can be seen as a prediction of how useful a neighbour's advice is expected to be for the target user, which has proved to be a quite effective approach. The question is whether other performance factors beyond user similarity can be considered in a way that further enhancements can be drawn, as research on user trust awareness has attempted to prove in the last years.

The Information Retrieval performance prediction view provides a methodological approach, which we propose to adapt to the neighbour selection problem. The approach provides a principled path to drive the formulation, development and evaluation of effective neighbour selection and weighting techniques, as we shall see. In the proposed view, the selection/weighting problem is expressed as an issue of neighbour performance, as an additional factor (besides user similarity) to automatically tune the neighbours' contribution to the recommendations, according to the expected goodness of their advice. As summarised in Section 5.1, there are three core concepts in the performance prediction problem as addressed in the Information Retrieval literature: performance predictor, retrieval quality assessment, and predictor quality assessment. Since we are dealing with the prediction of which users may perform better as neighbours, the above three concepts can respectively be translated into *neighbour performance predictor*, *neighbour quality*, and *neighbour predictor quality*. For the sake of simplicity, let us assume we can define a performance predictor as a function that receives as input a user profile  $u$  (in general, it could receive other users or items as well), the set of items  $I_u$  rated by that user, and the collection  $S$  of ratings and

items (or any other user preference and item description information) available in the system. Then, following the notation given used in Chapter 5, we define a neighbour performance prediction function as:

$$\hat{\mu}(u) \leftarrow \gamma(u, \mathcal{I}_u, S). \quad (8.4)$$

The function  $\gamma$  can be defined in different ways, for instance, by taking into account the rating distribution of each user, the number of ratings available in the system, and the (implicit or explicit) relations made by that user with the rest of the community. Essentially, the neighbour performance predictor is intended to estimate the true neighbour quality metric, denoted as  $\mu(u)$ , which is typically measured using groundtruth information about whether the neighbour's influence is positive. The application of this perspective is not trivial, and requires, in particular, a definition of what the performance of a neighbour means in this context – where no standard metric for neighbour performance is yet available in the literature.

Once the estimated neighbour performance prediction values  $\hat{\mu}(u_n)$  are computed for all users, the quality of the prediction can be measured as presented in Section 5.4.2, that is, either by measuring the correlation between the estimations and the real values  $\mu(u_n)$ , or by using classification accuracy metrics such as the F-measure. Since in this case we are interested in providing a ranking of users, this relates more with the traditional query performance task, and not with query difficulty (see Section 5.4.1), where the latter metrics are used. In other words, the neighbour predictor quality metric is defined as the following correlation:

$$q(\gamma) = \text{corr}([\hat{\mu}(u_1), \dots, \hat{\mu}(u_n)], [\mu(u_1), \dots, \mu(u_n)]). \quad (8.5)$$

Similarly to the situation in Information Retrieval, this correlation provides an assessment of the prediction accuracy (Carmel and Yom-Tov, 2010); the higher its (absolute) value, the higher the predictive power of  $\gamma$ . Moreover, the sign of  $q(\gamma)$  represents whether the two involved variables – neighbour prediction and neighbour quality – are directly or inversely correlated.

Besides validating any proposed predictor by checking the correlation between predicted outcomes and objective metrics, we may further test the effectiveness of the defined predictors by introducing and testing a dynamic variant of user-based collaborative filtering. In this variant, the weights of neighbours are dynamically adjusted based on their expected effectiveness, along with the decision of which users belong to each neighbourhood, as in the general formulation presented in Equation (8.2). We propose to define the neighbour scoring function  $s(u, v, i)$  based on the values computed from each neighbour performance predictors.

Hence, the basic idea of the framework presented here is to formally treat the neighbour selection and weighting in memory-based recommendation as a performance prediction problem. The performance prediction framework provides a principle basis to analyse whether the predictors are capturing some valuable, measurable

characteristic known to be useful for prediction, independently from their latter use in a recommendation strategy. Furthermore, if a neighbour scoring function with strong predictive power is introduced into the recommendation process and the performance is not improved, then, new ways of introducing such predictor into the rating estimation should be tested (either for selection or weighting), since we have some confidence that this function captures interesting user's characteristics, valuable for recommendation.

## 8.3 Neighbour quality metrics and performance predictors

The performance prediction research methodology requires a means to compare the predicted performance with the observed performance. This comparison is typically conducted in terms of some one-dimensional functional values, where the performance is assessed by some specific metric and the prediction can be translated to a certain numeric value. This value quantifies the expected degree of effectiveness, providing, thus, a relative magnitude.

Whereas in the context of performance prediction in IR, standard metrics of system effectiveness in response to a query are used for this purpose, in the case of predicting the performance of a neighbour for recommendation we would require to use metrics that measure how effective a neighbour is. In this section we propose several neighbour quality metrics and performance predictors which we shall evaluate in Section 8.4.

### 8.3.1 Neighbour quality metrics

The purpose of effectiveness predictors in our framework is to assess how useful specific neighbour profiles are as a basis for predicting ratings for the target user. Each predictor has to be contrasted to a measure of how “good” the neighbour's contribution is to the global community of users in the system. In contrast with query performance prediction, where a well established array of metrics are used to quantify query performance, to the best of our knowledge, in the literature there is not an equivalent function for neighbours used in user-based collaborative filtering. We therefore need to introduce and propose some sound candidate metrics.

Ideally, in the proposed framework, a quality metric should take the same arguments as the predictor, and thus, if we have, for instance, a user-item predictor, we should also be able to define a quality metric that depends on users and items. In general, we shall focus on user-based predictors, but it would be possible to explore item-based alternatives. Furthermore, we shall consider metrics taking neighbours as single input, independently from which neighbourhood is involved (i.e., independ-

ently from the target user), and which item is recommended. At the end of this section, nonetheless, we shall introduce a neighbour quality metric suitable for the user-user scenario, where both the target user and neighbour are taken into account.

Now, we propose three different neighbour quality metrics. The first two metrics had a different intended use by their authors, but we found they could be useful to evaluate how good a user is as a neighbour. The third metric was proposed by us in (Bellogín and Castells, 2010), where the problem of neighbour performance was explicitly addressed.

Rafter et al. (2009) propose two metrics in order to examine whether the neighbours have any influence in the recommendation accuracy. Both metrics are based on the comparison between true ratings and a neighbour's estimation of the ratings, as a way to measure the direction of the neighbour estimation and the average absolute magnitude of the shift produced by this estimation. Thus, the larger the neighbour's influence, the better her performance, according to our definition of a “good” neighbour. In this context we use those metrics as follows:

$$\mu_1 = \mu(v) = \frac{1}{|T_v|} \sum_{i \in T_v} \frac{1}{|N_k^{-1}(v; i)|} \sum_{w \in N_k^{-1}(v; i)} |r(w, i) - r(v, i)|$$

$$\mu_2 = \mu(v) = \frac{1}{|T_u|} \sum_{i \in T_v} \frac{1}{|N_k^{-1}(u; i)|} \sum_{w \in N_k^{-1}(v; i)} \delta([sgn(r(w, i) - \bar{r}(v)) = sgn(r(v, i) - \bar{r}(v))]; 1)$$

where  $\delta$  is a binary function whose output is 1 if its arguments are true, and 0 otherwise. Metric  $\mu_1$  represents the **absolute error deviation** of a particular user, and  $\mu_2$  is the **sign of error deviation**. Note that  $N_k^{-1}(v; i)$  denotes an inverse neighbourhood, which represents those users for whom  $v$  is a neighbour, and  $T_v$  denotes the items rated by user  $v$  in the test set. We can observe how each of these metrics represents a different method to measure how accurate the user  $v$  is as a neighbour.

In (Bellogín and Castells, 2010) we proposed a metric named **neighbour goodness**, which is defined as the difference in performance of the recommender system when including vs. excluding the user (i.e., her ratings) from the dataset. For instance, based on the mean average error standard metric, neighbour goodness can be instantiated as:

$$\mu_3 = \mu(v) = \frac{1}{|R_{U \setminus \{v\}}|} \sum_{w \in U \setminus \{v\}} [CE_{U \setminus \{v\}}(w) - CE_U(w)]$$

$$CE_X(v) = \sum_{i \in I, r(v, i) \neq \emptyset} |\tilde{r}_X(v, i) - r(v, i)|$$

where  $\tilde{r}_X(v, i)$  represents the predicted rating computed using only the data in  $X$ . This metric quantifies how much a user affects (contributes to or detracts from) the

total amount of mean average error of the system, since it is computed in the same way as that metric, but leaving out the user of interest – in the first term, the user is completely omitted; in the second term, the user is only involved as a neighbour. In this way we measure how a user contributes to the rest of users, or put informally, how better or worse the “world” is in the sense of how well recommendations work with and without the user. Hence, if the error increases when the user is removed from the dataset, it is considered as a good neighbour.

Based on the same idea of the previous metric, we propose a user-user quality metric that measures how one particular user affects to the error of another user when acting as her neighbour:

$$\mu_4 = \mu(u, v) = CE_{U \setminus \{v\}}(u) - CE_U(u)$$

We call this metric **user-neighbour goodness**. It quantifies the difference in user  $u$ 's error when neighbour  $v$  is not in the system against the error when such neighbour is present, that is, it measures how much each neighbour contributes to reduce the error of a particular user.

### 8.3.2 Neighbour performance predictors

Having formulated neighbour selection in memory-based recommendation as a task of neighbour effectiveness prediction, and having proposed effectiveness metrics to compare against, the core of an approach to this problem is the definition of effectiveness predictors. For this purpose, similarity functions and trust models such as those mentioned in Section 8.1 can be directly used, since in trust-aware recommendation, trust metrics aim at measuring how reliable a neighbour is when introduced in the recommendation process (O'Donovan and Smyth, 2005). Interestingly, some of them only depend on one user (**global trust metrics**), and others depend on a user and an item or another user (**local trust metrics**). Furthermore, other authors have proposed different indicators for selecting good neighbours, mainly based on the overlap between the user and her neighbour, without considering the concept of trust.

We thus distinguish three types of neighbour performance predictors: **user predictors** – equivalent to the global trust metrics –, **user-item predictors**, and **user-user predictors** – equivalent to the local trust metrics. Note that, although trust metrics could now be interpreted as neighbour performance predictors, the proposed performance prediction framework let us to provide an inherent value to these metrics (identified as performance predictors), independently from whether they improve a recommender's performance when used for selecting or weighting in the specific collaborative filtering algorithm. This is due to the fact that it is possible to empirically check the quality of the prediction by analysing their correlation with respect to the neighbour performance metric, prior to the integration in any collabora-

tive filtering method. Thus, each predictor would obtain an explicit score that represents its predictive power, related to our *a priori* confidence on whether such predictor is capturing the neighbour's reliability or trustworthiness.

In the following we propose an array of neighbour effectiveness prediction methods, by adapting and integrating trust functions from the literature into our framework, and we also propose novel prediction functions.

## User Predictors

User predictors are performance predictors that only depend on the target neighbour. When that neighbour is predicted to perform well, her assigned weight in the user-based collaborative filtering formulation is high.

One of the first user trust metrics proposed in the literature is the **profile-level trust** (O'Donovan and Smyth, 2005), which is defined as the percentage of correct recommendations in which a user has participated as a neighbour. If we denote the set of recommendations in which a user has been involved as

$$\text{RecSet}(u) = \{(v, i) : u \in N_k(v; i)\},$$

then the predictor is defined as follows:

$$\gamma_1(u, v, i) = \gamma(v) = \frac{|\text{CorrectSet}(v)|}{|\text{RecSet}(v)|},$$

where the definition of correct recommendations depends on a threshold  $\epsilon$ :

$$\begin{aligned} \text{CorrectSet}(u) &= \{(c_k, i_k) \in \text{RecSet}(u) : \text{Correct}(i_k, u, c_k; 1)\} \\ \text{Correct}(i, u, v; \lambda) &= \delta(|r(u, i) - r(v, i)| \leq \epsilon; \lambda), \end{aligned}$$

$\delta(a; b)$  being a binary function like before whose output is a value  $b$  if the predicate  $a$  is true, and 0 otherwise. That is, the recommendations considered as correct are those in which the user was involved as a neighbour, and her ratings were close (up to a distance of  $\epsilon$ ) to the actual ratings.

A similar trust metric, called **expertise trust**, is presented in (Kwon et al., 2009), where the concept of 'correct recommendation' is also used. In that work Kwon and colleagues introduce a compensation value for situations in which few raters are available. Specifically, the correct recommendation function only outputs a value of 1 when there are enough raters for a particular item (more than 10 in the paper). Otherwise, an attenuation factor is introduced by dividing the number of raters by 10, in the same way as significance weighting is introduced in Pearson's correlation in (Herlocker et al., 2002). More formally, the predictor is defined as:

$$\gamma_2(u, v, i) = \gamma(v) = \frac{1}{\sum_{j \in I_v} \sum_{w \in U_i} 1} \sum_{j \in I_v} \sum_{w \in U_i} \text{Correct}(j, v, w; \lambda(j))$$

where  $\lambda(j)$  is 1 when item  $j$  has more than 10 raters, and  $\mathcal{U}_i$  denotes the users who rated item  $i$ . In the same paper the authors propose another trust metric called **trustworthiness**, which is equivalent to the absolute value of the similarity between the target user's ratings and the average ratings given by the community (denoted as  $\bar{R}$ ). The authors introduce the significance weighting factor  $\beta$  as in (Herlocker et al., 2002), in a way that  $\beta(v)$  is 1 when user  $v$  has more than 50 ratings; otherwise,  $\beta$  is computed as the user's ratings divided by 50. Once the  $\beta$  factor is computed, the predictor is defined as follows:

$$\gamma_3(u, v, i) = \gamma(v) = \beta(v) \times \left| \frac{\sum_{j \in \mathcal{I}_v} (r(v, j) - \bar{r}(v))(\bar{r}(j) - \bar{R})}{\sqrt{\sum_{j \in \mathcal{I}_v} (r(v, j) - \bar{r}(v))^2 \sum_{j \in \mathcal{I}_v} (\bar{r}(j) - \bar{R})^2}} \right|$$

Hwang and Chen (2007) present a global trust metric, which we call **global trust deviation**, defined as an average of local (user-to-user) trust deviations. This metric makes use of the predicted rating for a user-item pair by using only one user as neighbour:

$$\tilde{r}(u, i) \sim \tilde{r}(u, i; v) = \bar{r}(u) + (r(v, i) - \bar{r}(v))$$

where user  $v$  is the considered neighbour. The predictor is then computed by averaging the prediction error of co-rated items between each user, and normalising the error according to the rating range  $R_r$  (e.g. in a typical 1 to 5 rating scale,  $R_r = 4$ ):

$$\gamma_4(u, v, i) = \gamma(v) = \frac{1}{|N_k(v)|} \sum_{w \in N(v)} \left( \frac{1}{|\mathcal{I}_v \cap \mathcal{I}_w|} \sum_{j \in \mathcal{I}_v \cap \mathcal{I}_w} \left[ 1 - \frac{|\tilde{r}(v, j; w) - r(v, j)|}{R_r} \right] \right).$$

Finally, a performance predictor inspired by the clarity score defined for query performance (Cronen-Townsend et al., 2002) was proposed in (Bellogín and Castells, 2010), considering its adaptation to predict neighbour performance in collaborative filtering. In the same way query clarity captures the lack of ambiguity in a query, **user clarity** is expected to capture the lack of ambiguity in a user's preferences. Thus, the amount of uncertainty involved in a user's profile is assumed to be a good predictor of her performance; and the larger the following value, the lower the uncertainty and the higher the expected performance:

$$\gamma_5(u, v, i) = \gamma(v) = KLD(v \parallel \mathcal{U} \setminus \{u\}) = \sum_{w \in \mathcal{U} \setminus \{v\}} p(w|v) \log_2 \frac{p(w|v)}{p(w)}$$

The probabilistic models defined in that work are based on smoothing estimations and conditional probabilities over users and items. Specifically, a uniform distribution is assumed for users and items, whereas the user-user probability is defined by an expansion through items as follows:

$$p(v|u) = \sum_{i \in \mathcal{I}_u} p(v|i)p(i|u).$$

Conditional probabilities are linearly smoothed with the user's probabilities and the maximum likelihood estimators, which finally depend on the rating given by the user towards an item; i.e.,  $p_{ml}(i|u) \propto r(u, i)$ .

It is interesting to note that this predictor (and the probability model in which is grounded) does not correspond with any of the adaptations of the clarity score proposed in Chapter 6, since relations between users are not considered in any of the rating-based probability models presented.

In addition to the integration of the above methods in the role of neighbour effectiveness predictors in our framework, we propose two novel predictors based on well known quantities measured over the probability models of (Bellogín and Castells, 2010): the entropy and the mutual information. Entropy, as an information-theoretic magnitude, measures the uncertainty associated with a probability distribution (Cover and Thomas, 1991). Borrowing the definition of user entropy from Chapter 6, we hypothesise that the uncertainty in the system's knowledge about a user's preferences may be a relevant signal in the effectiveness of a user as a potential neighbour, which could be captured by the entropy of the item distribution as follows:

$$\gamma_7(u, v, i) = \gamma(v) = -H(\mathcal{I}_v) = \sum_{j \in \mathcal{I}_v} p(j|v) \log_2 p(j|v).$$

Note that uncertainty, measured in this way, can be due to the system's knowledge about the user's tastes, or may come from the user herself (e.g. some users may have strong preferences, while others may be more undecided), and both causes may similarly affect the neighbour effectiveness. In either case the predictor can be interpreted as the lack of ambiguity in a user profile.

The second information-theoretic magnitude we propose to use over the probability models presented above is the mutual information. To be precise, the mutual information is a quantity computed between two random variables that measure the mutual dependence of the variables, or, in other terms, the reduction in uncertainty about one variable provided some knowledge about the other (Cover and Thomas, 1991). Here, we propose to adapt this concept, and compute the **mutual information** between the neighbour and the rest of the community in order to assess the uncertainty involved in the neighbour's preferences. For this purpose, instead of computing the mutual information over all the events in the sample space for both variables (users), we fix one of them (for the current neighbour), and move along the other dimension:

$$\gamma_6(u, v, i) = \gamma(v) = MI(v; \mathcal{U} \setminus \{u\}) = \sum_{w \in \mathcal{U} \setminus \{u\}} p(w|v) \log_2 \frac{p(w|v)}{p(v)p(w)}.$$

## User-Item Predictors

User-item predictors consist of performance predictors that depend on a user-item pair. More specifically, they are defined upon the active neighbour and the target item. This type of predictor is more difficult to apply because of its higher vulnerability to data sparsity. In a bi-dimensional user-item input space less observations can be associated to each input data point, whereby the confidence on the predictor outcome is lower, as it can be biased to outliers or unusual users or items.

A local trust metric based on the target user and item is proposed in (O'Donovan and Smyth, 2005). This metric is called **item-level trust**, and aims to discriminate reliable neighbours depending on the current item, since the same user may be more trustworthy for predicting ratings for certain items than for others. The formulation of this predictor can be seen as a particularisation of  $\gamma_1$ , but constraining the recommendation set only to the pairs in which the current item is involved:

$$\gamma_8(u, v, i) = \gamma(v, i) = \frac{|\{(c_k, i_k) \in \text{CorrectSet}(v) : i_k = i\}|}{|\{(c_k, i_k) \in \text{RecSet}(v) : i_k = i\}|}.$$

## User-User Predictors

The user-user predictors take as inputs two users: the active user and the current neighbour. User-user predictors based on local trust metrics have been studied further than user-item predictors in the literature, since the former are able to represent how much a user can be trusted by another, and let for different interpretations of the relation between users. These metrics have been often researched in the scope of social networks, and the users' explicit links in this context (Ziegler and Lausen, 2004; Massa and Avesani, 2007a), along with several trust metrics based on ratings, as we shall show below. In this way, although social-based metrics could be smoothly integrated in our framework, here we focus on a complementary view on trust where predictors are defined based on ratings. We leave other type of predictors as future work.

A first simple neighbour reliability criterion one may consider is the amount of common experience with the target user, that is, the amount of information upon which the two users can be compared. If we define "user experience" as the set of items the user has interacted with, we may define a predictor embodying this principle as:

$$\gamma_9(u, v, i) = \gamma(u, v) = |\mathcal{I}_u \cap \mathcal{I}_v|.$$

We shall refer to this predictor as **user overlap**. This predictor will serve as a basis for subsequent predictors, since most of them will depend on the items rated by both users. For instance, it has a clear use in assessing the reliability of the inter-user similarity assessments, which has been applied in the literature under a more practi-

cal, ad-hoc manner. Specifically, Herlocker et al. (2002) proposed the introduction of a weight on the similarity function, where the latter is devalued when it has been based on a small number of co-rated items. We may formulate **Herlocker's significance weighting predictor** as follows:

$$\gamma_{10}(u, v, i) = \gamma(u, v) = \frac{|\mathcal{I}_u \cap \mathcal{I}_v|}{n_H} \text{ if } |\mathcal{I}_u \cap \mathcal{I}_v| < n_H; 1 \text{ otherwise,}$$

where  $n_H$  is the minimum number of co-rated items that two users should have in common in order to avoid similarity penalisation. A value of  $n_H = 50$  was proved empirically to work effectively.

A variation of the previous scheme was proposed in (McLaughlin and Herlocker, 2004), to which we shall refer as **McLaughlin's significance weighting**:

$$\gamma_{11}(u, v, i) = \gamma(u, v) = \frac{\max(|\mathcal{I}_u \cap \mathcal{I}_v|, n_{Mc})}{n_{Mc}}.$$

This predictor is aimed to be equivalent to the Herlocker's significance weighting ( $\gamma_{10}$ ) formulation when  $n_{Mc} = n_H$ . However, we note that  $\gamma_{10}$  and  $\gamma_{11}$  represent different concepts, and are not fully equivalent. For instance, as noted in (Ma et al., 2007),  $\gamma_{11}$  may return values larger than 1 when  $|\mathcal{I}_u \cap \mathcal{I}_v| > n_{Mc}$ , while  $\gamma_{10}$ , by definition, always returns a value in the  $(0,1]$  interval.

Alternatively, the following variant can be drawn from (Ma et al., 2007), which is just a more compact reformulation of  $\gamma_{10}$ :

$$\gamma_{12}(u, v, i) = \gamma(u, v) = \frac{\min(|\mathcal{I}_u \cap \mathcal{I}_v|, n_M)}{n_M}.$$

A more elaborated predictor was proposed in (Weng et al., 2006). The rationale behind such predictor is to consider two situations depending whether or not user  $u$  takes into account the recommendation made by neighbour  $v$ . In this sense trustworthiness is defined as the reduction in the proportion of incorrect predictions of going from the latter situation to the former. The definition of this predictor, denoted as **user's trustworthiness**, is the following:

$$\gamma_{13}(u, v, i) = \gamma(u, v) = \frac{1}{|R|^2 - \sum_x n(u, v; x, \cdot)^2} \left[ |R| \sum_x \sum_y \frac{n(u, v; x, y)^2}{n(u, v; \cdot, y)} - \sum_x n(u, v; x, \cdot)^2 \right]$$

In this formulation  $|R|$  represents the number of allowed rating values in the system (e.g. in a 1 to 5 rating scale,  $|R| = 5$ ), the function  $n(u, v; x, y)$  represents the number of co-rated items on which  $v$ 's ratings have the value  $y$  while  $u$ 's ratings are  $x$ , that is,  $n(u, v; x, y) = |\{(u, \cdot, x)\} \cap \{(v, \cdot, y)\}|$  when each rating tuple is represented as  $(a, b, c)$ , given a user  $a$ , an item  $b$ , and a rating value  $c$ . In the same way,  $n(u, v; x, \cdot) = \sum_y n(u, v; x, y)$  represents all the co-rated items between  $u$  and  $v$ .

rated with any rating value by user  $v$ , and, analogously,  $n(u, v; \cdot, y) = \sum_x n(u, v; x, y)$ . In this case, the assumed hypothesis is that trust is one's expectation of other's competence in reducing its uncertainty in predicting new ratings.

Finally, a user-user predictor can be defined based on the global trust deviation predictor defined above ( $\gamma_4$ ). In fact, Hwang and Chen (2007) define **trust deviation** by ignoring the average along users as follows:

$$\gamma_{14}(u, v, i) = \gamma(u, v) = \frac{1}{|\mathcal{I}_u \cap \mathcal{I}_v|} \sum_{j \in \mathcal{I}_u \cap \mathcal{I}_v} \left[ 1 - \frac{|\tilde{r}(u, j; v) - r(u, j)|}{R_r} \right]$$

This predictor identifies effective neighbours mainly based on how many trustworthy (understood as “accurate”) recommendations a user has received from another.

## 8.4 Experimental results

In this section we report experiments in which the proposed neighbour effectiveness prediction framework is tested. First, we check the existing correlations between the user-based predictors defined in Section 8.3.2 and the neighbour performance metrics proposed in Section 8.3.1, as a direct test of their predictive power. For the user-item predictors we cannot analyse their correlation because we have no neighbour performance metric depending on both the target user and an item available.

Moreover, we test the usefulness of the predictors to enhance the final performance of memory-based algorithms, by using the predictors' values in the selection and weighting of neighbours, that is, by taking the predictors as the scoring function in Equation (8.2).

Our experiments were conducted on two versions of the MovieLens dataset, namely the 100K and 1M versions, described in Section 3.4.1 and Appendix A.1. For the user-based collaborative filtering method, we used Pearson's correlation as the similarity measure between users, and a varying neighbourhood size ( $k$ ), which is a parameter with respect to which the results were examined.

### 8.4.1 Correlation analysis

We analyse the correlation between neighbour quality metrics and neighbour performance predictors in terms of the Pearson and Spearman's correlation metrics. Correlation provides a measure of the predictive power of the neighbour effectiveness prediction approaches: the higher the (absolute) correlation value, the better the predictor estimates the positive neighbour effect on the recommendation accuracy. The sign of the correlation coefficient represents whether the two involved variables – neighbour quality metric and neighbour performance predictor – are directly or inversely correlated.

|                        | Absolute error deviation<br>$\mu_1 (-)$ | Neighbour goodness<br>$\mu_3 (+)$ | Sign of error<br>$\mu_2 (+)$ |
|------------------------|---|-----------------------------------|------------------------------|
| Clarity                | -0.21                                   | +0.17                             | +0.14                        |
| Entropy                | -0.18                                   | +0.18                             | +0.12                        |
| Expertise              | -0.62                                   | +0.03                             | +0.25                        |
| Global Trust Deviation | -0.35                                   | -0.01                             | +0.08                        |
| Mutual Information     | -0.20                                   | +0.17                             | +0.12                        |
| Profile Level Trust    | +0.62                                   | -0.04*                            | -0.24                        |
| Trustworthiness        | -0.21                                   | +0.03                             | +0.20                        |

**Table 8.1.** Pearson’s correlation between the proposed neighbour quality metrics and neighbour performance predictors in the MovieLens 100K dataset. Next to the metric name, an indication about the sign of the metric – direct(+) or inverse(-) – is included. Not significant values for a  $p$ -value of 0.05 are denoted with an asterisk (\*).

|                        | Absolute error deviation<br>$\mu_1 (-)$ | Neighbour goodness<br>$\mu_3 (+)$ | Sign of error<br>$\mu_2 (+)$ |
|------------------------|---|-----------------------------------|------------------------------|
| Clarity                | -0.30                                   | +0.16                             | +0.21                        |
| Entropy                | -0.22                                   | +0.17                             | +0.15                        |
| Expertise              | -0.65                                   | +0.02                             | +0.30                        |
| Global trust deviation | -0.38                                   | -0.03                             | +0.11                        |
| Mutual Information     | -0.25                                   | +0.16                             | +0.17                        |
| Profile Level Trust    | +0.65                                   | -0.02                             | -0.30                        |
| Trustworthiness        | -0.24                                   | +0.03                             | +0.25                        |

**Table 8.2.** Spearman’s correlation between quality metrics and performance predictors in the MovieLens 100K dataset.

Table 8.1 and Table 8.2 show the correlation values obtained on the MovieLens 100K dataset for the user-based predictors. We associate a sign to each quality metric indicating whether the metric is direct (denoted as '+') or inverse (denoted with '-'), according to the expected sign of the correlation with the predictor, i.e., a metric is direct if the higher its value, the better the true neighbour performance. We can observe that the Spearman’s correlation values are consistent, but slightly higher than Pearson’s, thus evidencing a non-linear relationship between the quality metrics and the performance predictors.

The absolute error deviation ( $\mu_1$ ) metric presents higher values when the neighbour’s prediction is less accurate, being thus an inverse neighbour metric. The other two metrics, sign of error ( $\mu_2$ ) and neighbour goodness ( $\mu_3$ ), are, by definition, direct neighbour metrics, since the former indicates how many times a recommendation from the neighbour has been made in the right direction, whereas the latter represents the change in error between excluding a particular user in the neighbourhood or including her, and thus, the larger this error, the “better” neighbour this user.

|                        | Absolute error deviation<br>$\mu_1 (-)$ | Neighbour goodness<br>$\mu_3 (+)$ | Sign of error<br>$\mu_2 (+)$ |
|------------------------|---|-----------------------------------|------------------------------|
| Clarity                | -0.14                                   | +0.40                             | +0.02                        |
| Entropy                | -0.07                                   | +0.39                             | -0.08                        |
| Expertise              | -0.95                                   | -0.06                             | +0.70                        |
| Global Trust Deviation | -0.55                                   | -0.24                             | +0.36                        |
| Mutual Information     | -0.17                                   | +0.30                             | +0.13                        |
| Profile Level Trust    | +0.83                                   | +0.04                             | -0.55                        |
| Trustworthiness        | -0.27                                   | +0.03                             | +0.36                        |

**Table 8.3. Pearson's correlation between quality metrics and performance predictors in the MovieLens 1M dataset. All the values are significant for a p-value of 0.05.**

|                        | Absolute error deviation<br>$\mu_1 (-)$ | Neighbour goodness<br>$\mu_3 (+)$ | Sign of error<br>$\mu_2 (+)$ |
|------------------------|---|-----------------------------------|------------------------------|
| Clarity                | -0.16                                   | +0.35                             | +0.04                        |
| Entropy                | -0.03                                   | +0.37                             | -0.10                        |
| Expertise              | -0.94                                   | -0.09                             | +0.69                        |
| Global trust deviation | -0.54                                   | -0.25                             | +0.39                        |
| Mutual information     | -0.16                                   | +0.31                             | +0.04                        |
| Profile level trust    | +0.94                                   | +0.09                             | -0.69                        |
| Trustworthiness        | -0.25                                   | +0.02                             | +0.37                        |

**Table 8.4. Spearman's correlation between quality metrics and predictors in the MovieLens 1M dataset.**

We can observe in Table 8.1 that, except for some of the predictors that obtain very low absolute values ( $< 0.10$ ), the four quality metrics are consistent with each other. This consistency is evidenced by the way the predictors correlate with the different metrics: some of the predictors obtain the correct correlations in every situation, that is, positive correlation with direct metrics and negative correlation with the inverse metric (like the clarity predictor), while other predictors obtain opposite values for all the metrics, that is, positive correlations with the inverse metric and negative correlations with direct metrics (such as the profile level trust predictor).

Also in Table 8.1 and Table 8.2 we see that each metric captures a different notion of neighbour quality because they show different correlation values with respect to the predictors. In this way, although consistent correlation results are obtained for direct and inverse metrics, each of them is actually detecting a different nuance of how a neighbour should behave in order to perform well.

Table 8.3 and Table 8.4 show the correlation values obtained on the Movie-Lens 1M dataset. We can observe that the trend in correlation is very similar to the behavior observed on the 100K dataset, and thus, similar conclusions can be drawn from it. There are, however, some changes in the absolute values of the correlation scores for some combinations of performance predictor and quality metric. For instance,

the clarity predictor and the neighbour goodness metric obtain larger values in this dataset, while the correlation between entropy and absolute error deviation is smaller.

It is important to note that the number of points used to compute the correlation values is different in the two datasets; there are less than 1,000 points in MovieLens 100K (with 943 users), and more than 6,000 points in MovieLens 1M dataset. This difference affects the significance of the correlation results, as already described in Section 5.4.2, where we observed how the confidence test for a Pearson's (and Spearman's) correlation depends on the size of the sample, and thus, the significance of a correlation value may change for different sample sizes.

In our experiments, for MovieLens 100K, the correlations are significant for a  $p$ -value of 0.05 when  $r > 0.05$ , and in the 1M dataset when  $r > 0.02$ . Hence, in Table 8.1, there is only one non-significant correlation value (denoted with an asterisk), whereas in Table 8.3, all the results are statistically significant.

Analysing in more detail the reported results for both datasets, we observe that the profile level trust predictor consistently obtains direct correlation values with inverse metrics, whereas inverse correlation values are obtained with direct metrics. This predictor seems to give higher scores to neighbours with larger deviations in their accuracy error, which would result on bad performance prediction because these values are not in the same direction than the performance metrics. The expertise and global trust deviation predictor obtain strong inverse correlations with the absolute error deviation metric, although their correlations with respect to the neighbour goodness metric are negligible, especially for the first predictor, in both datasets. At the other end of the spectrum, the clarity, entropy, and mutual information predictors obtain strong correlation values with the neighbour goodness, and moderate correlations with the rest of metrics, which make these predictors good candidates for successful neighbour performance predictors. Finally, the trustworthiness predictor obtains a significant amount of correlation with respect to the absolute error deviation and sign of error metrics, although its correlation with respect to the neighbour goodness is very low. This predictor thus seems to be useful on estimating how accurate the neighbour may be in terms of the error in a user basis, but probably not as a global metric.

Table 8.5 shows the correlations obtained for user-user neighbour predictors and the proposed user-neighbour clarity metric. Due to the high dimensionality of the vectors involved in this computation, we have considered only those users that have at least one item in common. Despite this fact, correlations are almost negligible, except for the McLaughlin's significance weighting predictor and the Spearman's coefficient, which evidences a non-linear relation between this predictor and the metric. In the next section we shall show that this function is one of the best performing predictors among the evaluated neighbour scoring functions. This result confirms the usefulness of the proposed neighbour performance metric since it is able to discrimi-

|                        | Movielens 100K |          | Movielens 1M |          |
|------------------------|----------------|----------|--------------|----------|
|                        | Pearson        | Spearman | Pearson      | Spearman |
| Herlocker              | 0.02           | 0.03     | 0.01         | 0.02     |
| McLaughlin             | 0.01           | 0.12     | 0.01         | 0.11     |
| Trust Deviation        | 0.01           | 0.01     | 0.01         | 0.01     |
| User Overlap           | 0.02           | 0.03     | 0.02         | 0.02     |
| User's Trustworthiness | -0.02          | -0.02    | -0.01        | -0.01    |

**Table 8.5. Correlation between the user-neighbour goodness and user-user predictors in the two datasets evaluated.**

nate which neighbour performance predictors are able to capture interesting properties between the user and her neighbours.

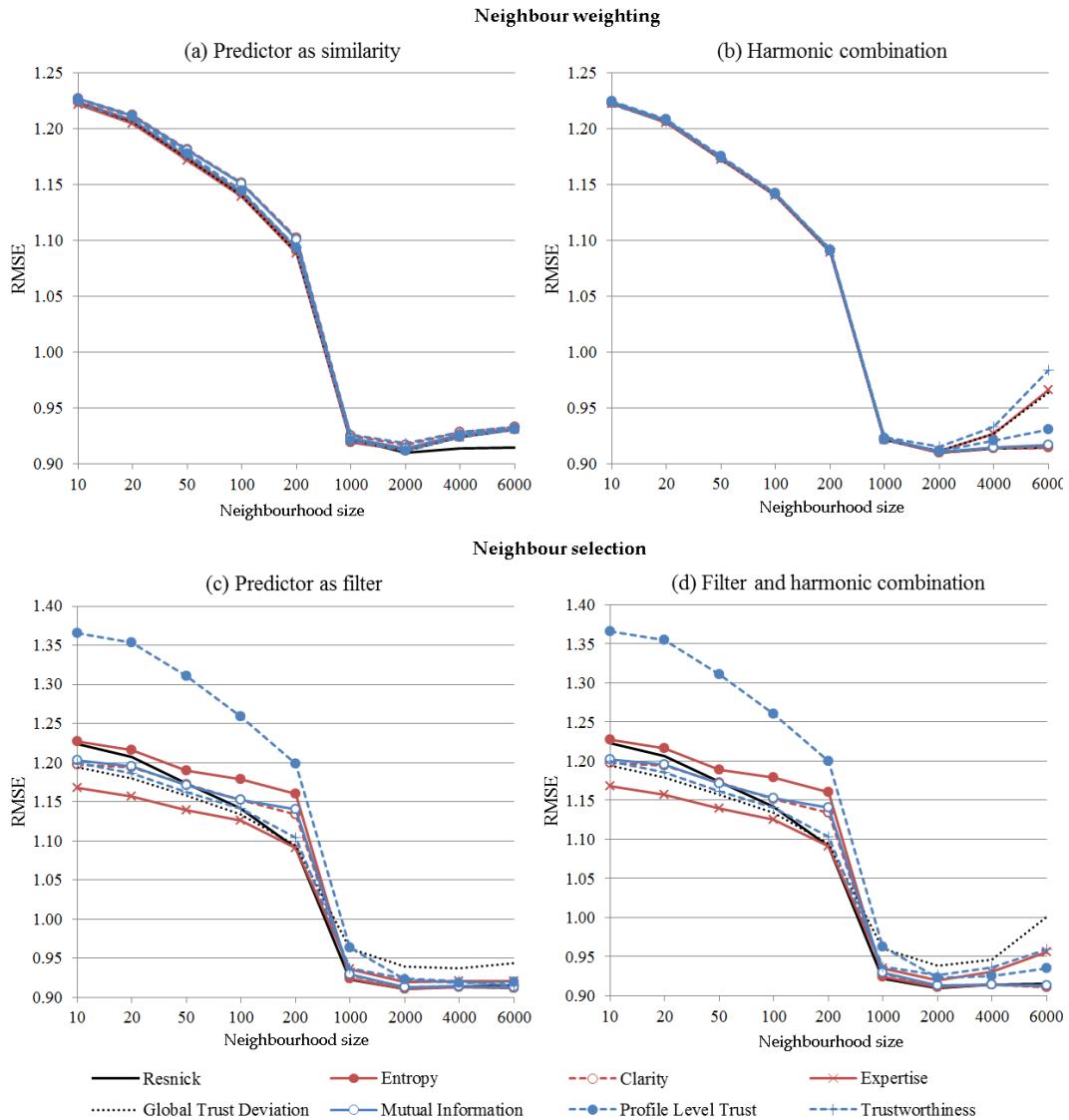
In summary, we have observed that most of the performance predictors agree with respect to the different performance metrics, and in general, the correlations computed between neighbour quality metrics and neighbour performance predictors are statistically significant.

#### 8.4.2 Performance analysis

The results reported in the previous section show that some of the studied predictors have the ability to capture neighbour performance, and because of that we hypothesise that they could be used to improve the accuracy of a recommendation model. This hypothesis, nonetheless, has to be checked since the metric against which we measure the neighbour goodness is not the same as the final recommendation performance metric we aim to optimise. With the experiments we report next we aim to confirm the usefulness of the proposed predictors, the validity of the proposed metrics as useful references to assess the power of the predictive methods, and the usefulness of the overall framework as a unified approach to enhance neighbourhood-based collaborative filtering.

In order to achieve this we test the integration of the neighbour predictors into a neighbour selection and weighting scheme for user-based collaborative filtering, as described in Section 8.2.1. Besides testing the effectiveness of the predictors, this experiment provides for observing to what extent the correlations obtained in the previous section correspond with improvements in the final performance of those predictors.

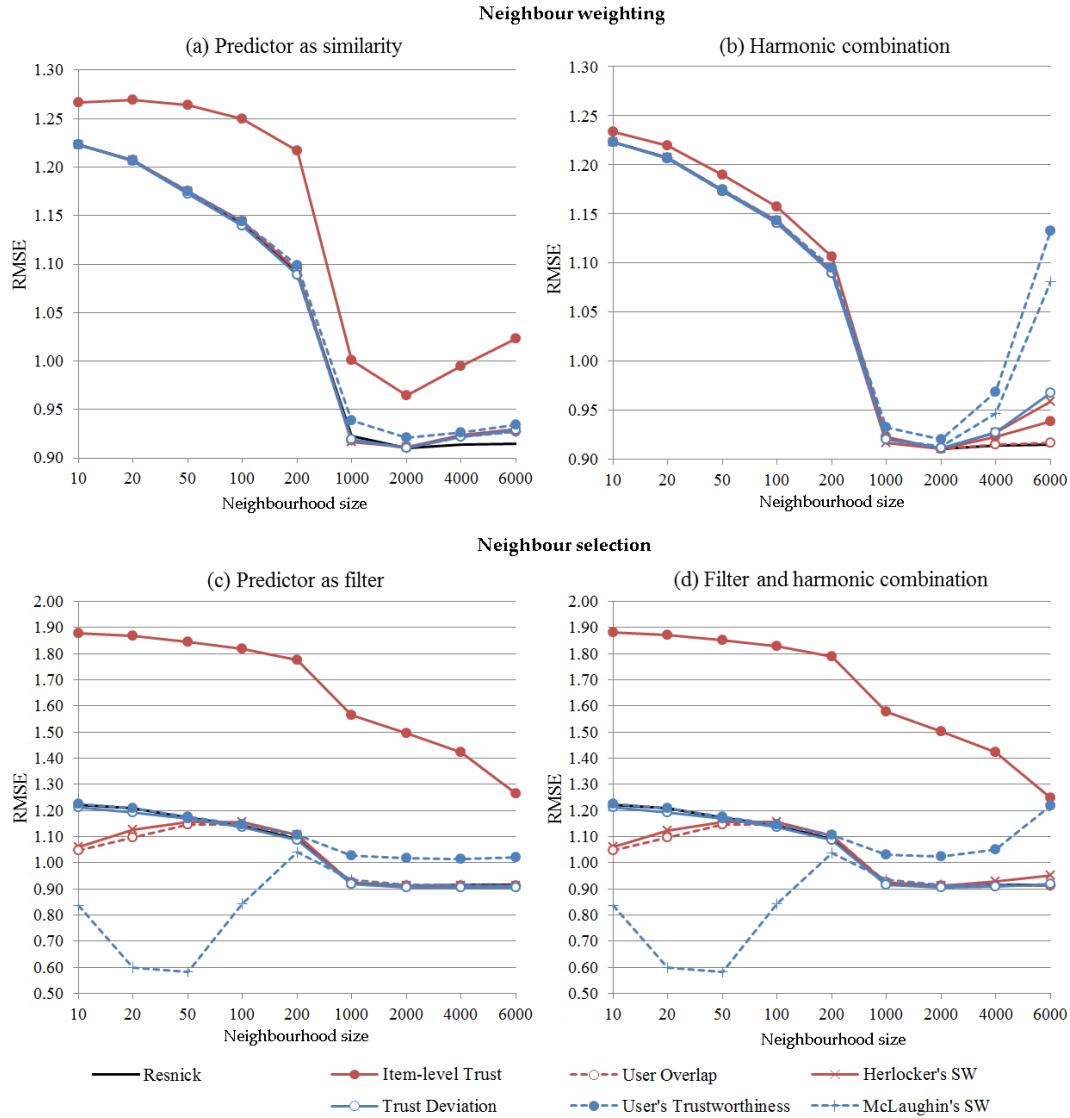
We provide recommendation accuracy and precision results on the MovieLens 1M dataset. Those obtained on the MovieLens 100K dataset are not reported here since they had similar trends. Figure 8.1 and Figure 8.2 show the Root Mean Square Error (RMSE) of the Resnick's collaborative filtering adaptation proposed in Equation (8.2) when used for different neighbour selection and weighting approaches. The curves at the top of the figures represent the values obtained when neighbour per-



**Figure 8.1. Performance comparison for user-based predictors and different neighbourhood sizes.**

formance predictors are used for neighbour weighting, that is, when the standard neighbour selection strategy is used ( $f^{neigh} = f_0^{neigh}$  in Equation (8.2)). Note that since the lines represent errors, the lower these values, the better the performance. Besides, Figure 8.3 presents the results found with the precision at 10 (P@10) ranking metric of a subset of the proposed methods, where in this case the higher the values, the better the performance.

A different aggregation function is used in each approach, depending on whether the harmonic mean between the predictor score and the similarity value (function  $f^{agg} = f_2^{agg}$ , on the right), or the projection function ( $f^{agg} = f_4^{agg}$ , on the left) are used, in the latter case in order to ignore the similarity. The curves at the bottom



**Figure 8.2. Performance comparison using user-item and user-user predictors for different neighbourhood sizes.**

of the figures show the neighbour selection approach ( $f^{neigh} = f_1^{neigh}$  in Equation (8.2)) along with the same neighbour weighting functions described above (i.e.,  $f_2^{agg}$  on the right and  $f_4^{agg}$  on the left). The rest of the aggregation functions, such as average ( $f_1^{agg}$ ) and product ( $f_3^{agg}$ ), were also evaluated for neighbour selection and weighting, but provided results equivalent to those of the harmonic mean. For this reason, they have been omitted in the figures to avoid cluttering them. We believe this equivalence may be due to the normalisation factor included in the collaborative filtering formulation, since it would cancel out the weights obtained by the harmonic, average, and product functions in the same way.

|                        | RMSE  |
|------------------------|-------|
| Resnick                | 1.174 |
| Clarity                | 1.181 |
| Entropy                | 1.175 |
| Expertise              | 1.171 |
| Global Trust Deviation | 1.173 |
| Mutual Information     | 1.180 |
| Profile Level Trust    | 1.177 |
| Trustworthiness        | 1.175 |

|                        | RMSE  |
|------------------------|-------|
| Resnick                | 1.174 |
| Herlocker              | 1.175 |
| Item-level Trust       | 1.264 |
| McLaughlin             | 1.174 |
| Trust Deviation        | 1.173 |
| User Overlap           | 1.175 |
| User's Trustworthiness | 1.175 |

**Table 8.6. Detail of the accuracy of baseline vs. recommendation using neighbour weighting; here, performance predictors are used as similarity scores (50 neighbours).**

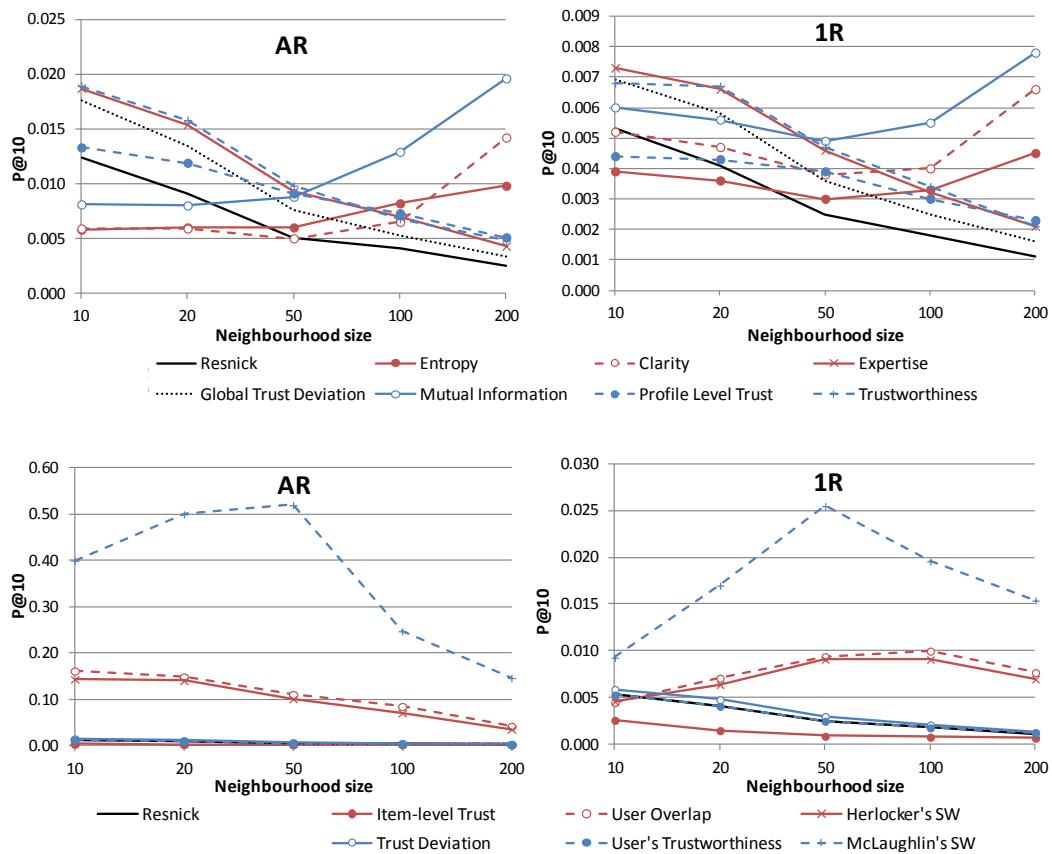
|                        | RMSE  |
|------------------------|-------|
| Resnick                | 1.174 |
| Clarity                | 1.172 |
| Entropy                | 1.189 |
| Expertise              | 1.139 |
| Global Trust Deviation | 1.158 |
| Mutual Information     | 1.171 |
| Profile Level Trust    | 1.310 |
| Trustworthiness        | 1.162 |

|                        | RMSE  |
|------------------------|-------|
| Resnick                | 1.174 |
| Herlocker              | 1.156 |
| Item-level Trust       | 1.843 |
| McLaughlin             | 0.581 |
| Trust Deviation        | 1.168 |
| User Overlap           | 1.146 |
| User's Trustworthiness | 1.174 |

**Table 8.7. Detail of the accuracy of baseline vs recommendation using neighbour selection; here, performance predictors are used for filtering (50 neighbours).**

Figure 8.1 shows the accuracy results when only user-based neighbour predictors are evaluated. We observe that, independently from the neighbourhood size, using performance predictors as similarity scores does not lead to large differences with respect to the baseline. These results are compatible with those presented in (Weng et al., 2006), where the improvement in RMSE is not very high ( $\Delta MAE < 0.05$  in that work). For the sake of clarity, in Table 8.6 and Table 8.7 we show the error values for a horizontal cut of the left curves; specifically, when the neighbourhood size is 50. We can observe that some predictors do improve Resnick's accuracy. Regarding the use of the harmonic mean as aggregation function (curves on the right), similar results are obtained except for very large neighbourhood sizes, for which some of the performance predictors produce worse results than the baseline, probably due to the amount of noise created by considering too many neighbours.

The curves at the bottom of the figures represent the accuracy results for neighbour selection strategies. In this case some of the predictors lead to worse performance than the baseline, particularly the profile level trust ( $\gamma_1$ ). This situation is consistent with the correlations observed in the previous section, since this predictor obtained inverse correlations with the different metrics, i.e., direct correlation values



**Figure 8.3. Performance comparison using ranking-based metrics for both user and user-user neighbour predictors using the AR and 1R evaluation methodologies.**

with inverse metrics, and inverse values with direct metrics. Moreover, as predicted by the correlation analysis, trustworthiness ( $\gamma_3$ ), mutual information ( $\gamma_6$ ), and clarity ( $\gamma_5$ ) result in some of the best performing recommenders (with strong correlations), as shown in the figures and in Table 8.7, along with expertise ( $\gamma_2$ ) and global trust deviation ( $\gamma_4$ ), which obtained more moderated correlation values.

In Figure 8.2 we can see how user-item and user-user neighbour predictors affect the performance of collaborative filtering recommenders. The curves in the top show that most of the predictors obtain a similar performance to that of the baseline, except for the item-level trust ( $\gamma_8$ ), the performance of which is much worse than Resnick's. Table 8.6 shows the specific error values for these recommenders. It is interesting to note that the performance of this predictor is drastically improved when using the harmonic mean as the aggregation function (shown on the right side of the figure). Similarly to user-based neighbour predictors (Figure 8.1), some of the user-item and user-user predictors decrease their accuracy with large neighbourhoods; in this case, user's trustworthiness ( $\gamma_{13}$ ) and McLaughlin's significance weighting ( $\gamma_{12}$ ) are the more representative examples.

A different conclusion results when neighbour selection is analysed (curves at the bottom). Two of the predictors are characterised by a much better (McLaughlin's significance weighting,  $\gamma_{12}$ ) or worse (item-level trust,  $\gamma_8$ ) final performance, independently from the weighting aggregation function. Table 8.7 shows the specific error values obtained for each of these predictors. It is interesting how the McLaughlin's predictor, despite its inability to boost good neighbours (see top figures), seems to be very useful for neighbour selection. This effect, nonetheless, is attenuated when the neighbourhood increases, since in that situation, selection methods have to deal with too many users in each neighbourhood. We believe the reason why this predictor is very good for neighbour selection is because it gives higher scores to those neighbours that have more items in common with the target user, and thus the confidence in the computation of the similarity values between the neighbour and the target user is higher. It is worth noting that, to the best of our knowledge, this function has never been used for neighbour selection, since its original motivation was to penalise the similarity value whenever it has been based on a small number of co-rated items. However, by plugging this function into our framework, and measuring its predictive power for user-neighbour performance, a novel application naturally emerges and provides very good results.

Finally, in Figure 8.3 we can observe that a similar trend is found with P@10 for both user-based predictors (top curves), and user-item and user-user predictors (bottom curves). In the figure we only present the results of the neighbour selection and weighting approaches for less than 200 neighbours, since the results of the rest of the approaches and neighbourhoods are very similar. It is worth noting that the two methodologies evaluated – AR and 1R – agree on the order of the best and worst performing dynamic approaches, although as already observed in the previous chapter, the absolute performance values obtained with each methodology may be very different – e.g. the maximum P@10 value with 1R is 0.1, which is reached by several recommendation methods with the AR methodology. More interestingly, these results show consistency between the performance of some dynamic approaches using error- and ranking-based metrics, since the best and worst predictors according to RMSE and P@10 are the same; McLaughlin's significance weighting and item-level trust, respectively. Moreover, the entropy and clarity user-based predictors show worse performance in small neighbourhoods, but outperform the baseline significantly in larger neighbourhoods, something different to what we observed in the previous experiment with error-based metrics.

In summary, we have been able to validate both the proposed user-user neighbour performance metrics, and the different evaluated user-user neighbour performance predictors. We have obtained positive results when this type of predictors has been introduced and compared against the baseline in the different aggregation strategies and configurations, and these results are consistent with the correlations

obtained between the predictors and the performance metrics. In particular, McLaughlin's significance weighting obtains an improvement up to 55% in both accuracy (i.e., error decrease) and precision (i.e., precision improvement) when this predictor is used to select the neighbours which will further contribute to the rating prediction. Besides, the (Spearman's) correlation for this predictor is positive and strong, in contrast to the values obtained for the rest of user-user predictors, which did not improve the accuracy of the baseline. In this context, a possible drawback of the conducted analysis is that we have not been able to define neighbour performance metrics based on user-item pairs, and thus the user-item neighbour performance predictors are out of the scope of the developed correlation analysis. Nevertheless, the obtained results showed that the only user-item neighbour performance predictor defined here – the item-level trust – is not able to outperform the baseline recommender. We believe this fact, which is in contradiction with what was reported in (O'Donovan and Smyth, 2005), may be caused by the different variables taking place in our evaluation, such as the dataset (MovieLens 1M instead of MovieLens 100K), the neighbourhood size (not specified in the original paper), and the several aggregation functions and combinations used across our experiments.

### 8.4.3 Discussion

The reported experiment results provide empiric evidence of the usefulness of the proposed framework, and the specific proposed predictors, as an effective approach to enhance the accuracy of memory-based collaborative filtering. As described in the preceding sections, the methodology comprises two steps, one in which the predictive power of neighbour predictors is assessed, and one in which the predictors are introduced in the collaborative filtering scheme to enhance the effectiveness of the latter. Our experiments confirm a strong correlation for some of the predictors – both user predictors and user-user predictors –, and this has been found to correspond with final accuracy enhancements in the recommendation strategy: the predictors that obtain strong direct correlations with the performance metrics are the best performing dynamic strategies; the profile level trust predictor, which obtains inverse correlation values with respect to the neighbour performance metrics, is the worst performing dynamic strategy.

In light of these results, it could be further investigated whether the actual correlation values between neighbour performance predictors and neighbour performance metrics could be used to infer how each predictor should be incorporated into a memory-based collaborative filtering method as a neighbour scoring function, since there is no obvious link between the ranking of the best performing scoring functions and the strength of their corresponding correlations. As a starting point, only the sign of the correlation could be considered, using either the raw neighbour predictor score (for positive correlations) or its inverse (for negative values). Then, this

rationale could be further elaborated and evaluated in order to check whether the performance improvements are consistent.

Research on finding functions with strong correlation power with respect to neighbour performance metrics could be an interesting area by itself, since it could have different final applications. We have experimented here with variations in neighbour selection and weighting for user-based collaborative filtering, but those predictors (functions) could also be used, for instance, for active learning (Elahi, 2011), or for providing more meaningful explanations (Marx et al., 2010), depending or based on the predicted performance of a particular user’s neighbours.

## 8.5 Conclusions

We have shown in this chapter that performance prediction does not only serve to aggregate entire recommender systems, but also to aggregate subcomponents of recommender algorithms – in this case, neighbour related terms in collaborative filtering. We propose a theoretical framework for neighbour selection and weighting in user-based recommender systems, which is based on a performance prediction approach drawn from the query performance methodology of the Information Retrieval field. By viewing the neighbourhood-based collaborative filtering rating prediction task as a case of dynamic output aggregation, our approach places user-based collaborative filtering in a more general frame, linking to the principles underlying the formation of ensemble recommenders, and rank aggregation in Information Retrieval. By doing so, it is possible to draw concepts and techniques from these areas, and vice versa. Our study thus provides a comparison of different state-of-the-art rating-based trust metrics and other neighbour scoring techniques, interpreted as neighbour performance predictors, and evaluated under this new angle. The framework lets an objective analysis of the predictive power of several neighbour scoring functions, integrating different notions of neighbour performance into a unified view. Thus, the proposed methodology discriminates which neighbour scoring functions are more effective in predicting the goodness of a neighbour, and thus identifies which weighting functions are more effective in a user-based collaborative filtering algorithm.

Drawing from different state-of-the-art neighbour scoring functions – cast as user, user-user, and user-item neighbour performance predictors –, we have reported several experiments in order to, first, check the predictive power of these functions, and second, validate them by comparing the final performance of neighbour-scoring powered memory-based strategies with that of the standard collaborative filtering algorithm. We also evaluate different ways to introduce these functions in the rating prediction formulation, namely for neighbour weighting, neighbour selection, and combinations thereof. In this context, methods where neighbour scoring functions

were integrated outperform the baseline for different values of neighbourhood size and predictor type.

We have also proposed several neighbour performance metrics that capture different notions of neighbour quality. The evaluated performance predictors show consistent correlations with respect to these metrics, and some of them present particularly strong correlations. Interestingly, a correspondence is confirmed between the correlation analysis and the final performance results, in the sense that the correlation values obtained between neighbour performance predictors and neighbour performance metrics anticipate which predictors will perform better when introduced in a memory-based collaborative filtering algorithm.

This research opens up the possibility to several research lines for the integration of other types of predictors and trust metrics into our framework. For instance, performance predictors defined upon social data, such as those defined in Chapter 6 based on user's trust network, could be smoothly integrated into our framework and analysed in the future. Furthermore, alternative neighbour performance metrics may be defined to check the predictive power of user-user and user-item predictors. These metrics may help better understand which characteristics of the neighbour performance such predictors are capturing, although based on a smaller amount of information since in rating-based systems users only rate items once. In particular, our framework would allow for different interpretations of the user's performance, by modelling different neighbour performance metrics, which may be oriented to accuracy (using error metrics as in this chapter), ranking precision, or even alternative metrics such as diversity, coverage and serendipity (Shani and Gunawardana, 2011). Additionally, other predictors based on item information could be defined similar to those proposed in (Weng et al., 2006; Ma et al., 2007), and easily incorporated into our framework using item-based algorithms instead of user-based.



# Part V

## Conclusions

*Not everything that can be counted counts,  
and not everything that counts can be counted.*

Albert Einstein



# Chapter 9

## Conclusions and future work

In this thesis we have investigated how to measure and predict the performance of recommender systems. We have analysed and proposed an array of methods based on the adaptation of performance predictors from Information Retrieval – mainly the query clarity predictor, which captures the ambiguity of a query with respect to a given document collection. We have defined several language models according to various probability spaces to capture different aspects of the users and items involved in recommendation tasks. In this context, we have proposed and evaluated novel approaches drawing from Information Theory and Social Graph Theory for different recommender input spaces, using information-theoretic properties of the user's preferences and graph metrics such as PageRank over the user's social network.

Moreover, since we aimed to predict the performance of a particular recommender system, we required a clear recommender evaluation methodology against which performance predictions can be contrasted. Hence, in this thesis we addressed the evaluation methodology as part of the problem, where we have identified statistical biases in the recommendation evaluation – namely the sparsity and popularity biases – which may distort the performance assessments, and therefore may confound the apparent power of performance prediction methods. We have analysed in depth the effect of such biases, and have proposed two experimental designs that are able to neutralise the popularity bias: a percentile-based approach and a uniform-test approach. The systematic analysis of the evaluation methodologies and the new proposed variants have enabled a more complete and precise assessment of the effectiveness of our performance prediction methods.

On the other hand, we have exploited the proposed performance prediction methods in two applications where they are used to dynamically weight different components of a recommender system, namely the dynamic adjustment of weighted hybrid recommendations, and the dynamic weighting of neighbours' preferences in user-based collaborative filtering. Through a series of empirical experiments on several datasets and experimental designs, we have found a correspondence between the predictive power of our performance predictors and performance enhancements in the two tested applications.

In this chapter we present the main conclusions obtained in our research work. In Section 9.1 we provide a summary and a discussion of our contributions, and in Section 9.2 we provide research directions that could be addressed in future work.

## 9.1 Summary and discussion of contributions

In the next subsections we summarise and discuss the main contributions of this thesis, addressing the research goals stated in Chapter 1. These contributions are organised according to the three main objectives addressed. First, we analysed how to properly evaluate recommender systems in order to obtain unbiased measurements of a recommender system's performance. Second, we proposed performance predictors that aim to estimate the performance of a recommendation method. And third, we used our performance predictors to dynamically combine components of a recommender system.

### 9.1.1 Analysis of the definition and evaluation of performance in recommender systems

We have analysed different experimental designs existing in the literature about recommender systems, oriented in particular to ranking-based evaluation, and have shown that **assumptions and conditions underlying the Cranfield paradigm are not granted in usual recommendation settings**. Specifically, we have detected statistical confounders (biases) that arise in applying that paradigm to the evaluation of recommender systems. We have shown that the specific value of the evaluation metric has a use for comparative purposes, but has no particular absolute meaning by itself. We have shown that precision decreases linearly with the sparsity of relevant items (**sparsity bias**) in the AR evaluation methodology, whereas it does not suffer from such bias in the 1R approach.

We have also observed that a non-personalised recommender based on item popularity obtains high performance values, and have shown and analysed in detail how this is due to a **popularity bias** in the experimental methodology. To address these issues, we have proposed **novel experimental approaches that effectively neutralise the popularity bias**.

### 9.1.2 Definitions and adaptations of performance predictors for recommender systems

We have defined and elaborated **performance predictors in the context of recommendation**, usually taking the user as the object of the prediction, but also considering items as an alternative prediction input. Specifically, we have adapted the query performance predictor known as *query clarity* by taking different assumptions and formulations into several variations of **user clarity** predictors. We have also used information theoretical related concepts such as entropy, graph metrics like centrality, PageRank, and HITS, and other domain-specific, heuristic approaches. We have

defined these predictors upon three input spaces of user preferences: **ratings, logs, and social networks**. On ratings and logs we have defined several language models and vocabulary spaces in such a way that our adaptations of clarity would capture different aspects of the user in a unified formulation for both input spaces. Within the same framework, we have introduced the temporal dimension on log-based preference data, drawing and elaborating time-based performance predictors proposed in prior work in the IR field for ad-hoc search.

Additionally, we have defined **item-based predictors** when rating-based preferences are used, which aim to estimate the performance of the items under consideration (to be more precise, the performance of a recommender system in suggesting those items). Here, the main problem is how to define the true performance metric that the predictor is aimed to estimate, since the items are not the main input of the recommendation process. For this reason, we have developed novel methodologies where the performance of an item can be measured, also considering possible biases arising from heavy raters that may distort the results just for statistical reasons.

We have assessed the predictive accuracy of our methods by computing the correlation between estimated and true performance, following standard practice in the IR performance prediction literature. In doing so, we used the unbiased methodologies analysed throughout the thesis to **compare how the predictors behave when the sparsity and popularity biases have been neutralised**. We have found strong correlation values confirming that our approaches result in a **significant predictive power**.

### 9.1.3 Dynamic weighting in recommender ensembles

Prevalent in the Recommender Systems literature we find combination of recommenders into the so-called recommender ensembles, which are a special type of hybrid recommendation methods where several recommenders are combined, and which are currently very common in the field as represented by current competitions (Bennett and Lanning, 2007; Dror et al., 2012). Collaborative Filtering, one of the major techniques used among the array of available recommendation strategies, can also be seen as a combination of several utility subfunctions, each corresponding to one neighbour (in user-based CF). In the same way performance prediction in Information Retrieval has been used to optimise rank aggregation, we have investigated the use of recommendation performance predictors to dynamically aggregate the output of recommenders and neighbours.

We have defined a **dynamic hybrid framework** where recommender ensembles can benefit from dynamic weights according to performance predictors with which strong correlations have been found. Our results indicate that high correlation with performance tends to correspond with enhancements in dynamic hybrid recommenders. Additionally, dynamic ensembles of recommenders usually outperform

baseline static ensembles for different recommender combinations and the three types of performance predictors investigated.

On the other hand, we have also proposed a **framework for neighbour selection and weighting** in user-based recommender systems. We have defined neighbour performance predictors and metrics by adapting and integrating some of the methods from the trust-aware recommendation literature. Our framework unifies several notions of neighbour performance under the same view, and provides an objective analysis of the predictive power of different neighbour scoring functions. Once the predictive power of these neighbour predictors was confirmed, we used them to weight the information coming from each neighbour in a dynamic fashion, by means of different strategies that combine similarity values and neighbours' weights. Our experiments confirm a correspondence between the correlation analysis and the final performance results, in the sense that the correlation values obtained between neighbour performance predictors and neighbour performance metrics anticipate which predictors will perform better when introduced into the user-based collaborative filtering algorithm.

## 9.2 Future work

Performance prediction in recommendation is an interesting research topic also from a business perspective, since one could decide when to deliver certain item recommendations to a user, avoiding lowering the user's confidence on the relevance of the recommendations. In this sense, performance predictions of potential recommendations may give control to the service provider; a control that could be used in various ways, such as recommendation combination methods more general than those addressed in this thesis. Regardless of the plausible applications for industry, and beyond the achievements presented throughout the thesis, we envision the following potential future research lines.

The evaluation of recommender systems still is an object of active research in the field, where several questions need more attention, such as the gap between offline and online experiments, and the missing not at random assumption. Nonetheless, in this thesis we have focused our research on aspects related to the prediction of performance, which requires a deeper understanding of the evaluation methodologies used. In this way, we could **extend our analysis of evaluation methodologies to other ranking metrics** such as those based on two rankings (NDPM, and Spearman's and Kendall's correlations) and those adapted from Machine Learning (e.g. AUC). In this way, we may find that one of these metrics is not influenced by any of the confounders described in Chapter 4, or that none of the design alternatives proposed are able to neutralise these effects. As an example of the interest of this topic, recently in (Pradel et al., 2012) the authors analysed the popularity effects over the

AUC metric and found that considering missing data as a form of negative feedback during training may improve performance, although it may also favour popularity-based recommenders over personalised recommendation methods.

Additionally, it would be beneficial for our research to be able to validate the usefulness of the unbiased measurement of performance with **online evaluations**. This would be valuable for a comparative assessment of the offline observations along with a deeper understanding of the extent to which popularity may be or not a noisy signal. Such a user study would help us determine the real benefits (if any) of receiving popular recommendations, since, for instance, by definition these suggestions are not novel, and probably neither serendipitous nor diverse.

In Chapter 6 we have proposed several performance predictors for recommendation based on the same principles as those denoted in IR as pre-retrieval predictors, like the clarity score, where the output of the retrieval engine (or the recommender system in our case) is not used by the predictor. Based on our results, the research possibilities to investigate more performance predictors for recommendation are abundant. In this line, several authors have exploited the **combination of predictors to obtain higher correlation values and stronger predictive power**, such as (Hauff et al., 2009) and (Jones and Diaz, 2007), where penalised regression and linear regression followed by neural network learning were used respectively. In those works the combination of predictors from different nature improved the correlation against the target evaluation metric – i.e., average precision. Thus, we envision the combination of predictors as a worthwhile direction also for recommendation, especially since we have defined predictors based on different inputs that are expected to have low redundancy between them and, when possible, the combination of such predictors may produce higher correlations for different types of inputs. Examples of these combinations may be the mixture of social and temporal dimensions, item-based temporal predictors, and other contextual dimensions not addressed in this thesis.

Moreover, a future investigation could **analyse and adapt to recommender systems post-retrieval performance predictors** defined in the IR literature, such as those based on the analysis of the score distribution from the recommended items to each user. This may provide predictors with stronger correlations and, thus, with more predictive power of the recommenders' performance, as it occurs in IR where post-retrieval predictors usually obtain higher correlation values than pre-retrieval predictors. The main limitation of this type of predictors is that they cannot be used directly to adapt the output of the recommender, since the complete output – i.e., the ranking – is typically required for the computation of the predictor values. This would require thinking of different applications where this type of predictors could be applied to recommendation.

A particular direction worth considering, and also related to Chapter 6, would be the **use of alternative evaluation approaches** beyond correlation metrics, such as those based on clustering the true and estimated performance values (see Section 5.4.2). In our work we have focused on the use of correlation metrics, mainly Pearson's correlation. These metrics have well-known limitations, such as their sensitivity to outliers, and the small (not significative) differences in correlation when a small number of points is used. For this reason, other approaches for assessing the predictive power of the predictors have been proposed. We have to note, however, that the use of a particular evaluation technique should be focused on their application to specific contexts (Pérez-Iglesias and Araujo, 2010); specifically, this requires defining new applications for performance predictors that match the evaluation metric, which we also envision as a potential future work.

Besides, in the same chapter we developed an evaluation methodology to assess the true performance values of the items, in order to evaluate the proposed item predictors. This methodology should be further validated in order to **obtain a fair measure of item performance**, which at this moment is still an open problem. In that way we would be able to define additional item predictors for other input spaces apart from ratings, and improve the predictiveness of the current item performance predictors.

In Chapter 7 we presented experiments regarding the dynamic combination of recommenders in an ensemble. Those experiments were limited to only one performance predictor for a pair of recommenders. We plan to extend these experiments with **ensembles where two predictors are considered** in order to investigate which conditions should be fulfilled by each pair of predictors in order to improve the performance of the ensemble. A related research direction worth of consideration would be the analysis of the sensitivity of the correlation values for which good performance results are obtained in the dynamic hybrid methods. More specifically, we may consider whether it is better to have an overall strong correlation value (in average) or a not very strong average correlation but better estimates for some particular users, where these users would play a significant role in the system such as the power users defined in (Lathia et al., 2008). A study like the one presented in (Hauff et al., 2010) could then be conducted where simulations of predictors with different correlations are evaluated and their effect on the final performance of the ensembles is compared against each other.

Furthermore, another limitation of the experiments presented in Chapter 7 was that the size of ensembles was always two. We aim to consider **ensembles of N recommenders** and, eventually as mentioned above, using one performance predictor for each recommender. This is a natural but non-trivial step towards a generalisation of the proposed framework to larger ensemble recommenders. Alternatively, Machine Learning techniques could be used to learn the best weights to use in the en-

semble in a user and item basis. In this case, a compromise between the computational costs of each technique (machine learning against performance predictors), their predictive power and the tendency to overfitting should be investigated.

Finally, in Chapter 8 we investigated the dynamic neighbour weighting problem using neighbour performance predictors oriented to error-based metrics. The future work related to this chapter could focus on the **adaptation of the neighbour performance metrics used in our approach to ranking-based metrics**, such as precision and recall. As we have already discussed, error metrics are not the best way to measure performance, although they can be considered appropriate in this context since we want to measure the improvement in accuracy of our approaches, along with facilitating comparisons with the state of the art in trust-aware recommendation, where these metrics are prevalent. Therefore, the use of ranking metrics would be a valuable contribution to the field by itself. Furthermore, once a neighbour performance metric based on a ranking metric is provided, we would be able to measure the correlation of the neighbour predictors described in that chapter with such metric, and analyse in detail the predictive power of predictors for ranking metrics. Ideally, we would be able to obtain a predictor with enough predictive power using both types of neighbour performance metrics (based on error and ranking), although this is not easy to grant in general, since each metric is defined to optimise different parameters and concepts.



# Part VI

# Appendices

*An algorithm must be seen to be believed.*

Donald Knuth



# **Appendix A**

## **Materials and methods**

In this thesis we have reported results obtained in experiments conducted with a variety of input data sources and recommendation algorithms. In this appendix we provide additional details, not reported in previous chapters, about such datasets and recommenders. Hence, in Section A.1 we present statistics about the datasets, and in Section A.2 we describe the specific configuration setting of the recommenders. Next, in Section A.3 we detail the followed evaluation methodologies. Finally, in Sections A.4 and A.5 we present further results regarding prediction correlations and performance of dynamic recommender ensembles by means of metrics such as MAP@10 and nDCG@50. Despite the fact the thesis is mainly focused on ranking metrics, for the sake of brevity and clarity, in Chapters 6 and 7 we only reported experimental results based on the P@10 metric.

## A.1 Datasets

In Section 3.4 we presented the three datasets used in this thesis, namely MovieLens, Last.fm, and CAMRa datasets. We now describe the specific partitions in which we splitted those datasets for experimentation. Specifically, we explain how the data splits were generated, and provide some statistics of the splits, such as their number of users and items, and their density, measured as the percentage of cells in the ratings matrix with known values, i.e.,  $\# \text{ratings} / (\# \text{users} \times \# \text{items}) \times 100$ .

### A.1.1 MovieLens dataset

In addition to the well-known Netflix dataset, the MovieLens 100K and MovieLens 1M datasets – extracted from the MovieLens movie recommendation system by the GroupLens research group at University of Minnesota, USA – have been the most widely used datasets in the Recommender Systems field. The former has 943 users, 1,682 items, and 100,000 ratings, whereas the latter has 6,040 users, 3,900 items, and 1 million ratings. In this thesis, when we do not explicitly indicate which of the two datasets was used, we refer to the MovieLens 1M dataset.

In the experiments we always performed a 5-fold cross validation strategy to generate 5 random 80-20% disjoint splits of rating sets. As rating sets, in the **MovieLens 100K** dataset we used the partition provided in its public distribution, and in the **MovieLens 1M** dataset we used 5 splits with 200,000 ratings, each of them randomly selected.

| Property                      | Overall | Average                      | Average<br>in training   | Average<br>in test        |
|-------------------------------|---------|------------------------------|--------------------------|---------------------------|
| No. users                     | 943     | 854.60<br>( $\pm$ 165.44)    | 943<br>( $\pm$ 0.00)     | 766.20<br>( $\pm$ 205.06) |
| No. items                     | 1,682   | 1,531.20<br>( $\pm$ 127.18)  | 1651.60<br>( $\pm$ 4.77) | 1410.80<br>( $\pm$ 11.52) |
| No. ratings                   | 100,000 | 50,000<br>( $\pm$ 31,622.78) | 80,000<br>( $\pm$ 0.00)  | 20,000<br>( $\pm$ 0.00)   |
| Density                       | 6.30%   | 3.56%<br>( $\pm$ 1.72)       | 5.14%<br>( $\pm$ 0.01)   | 1.99%<br>( $\pm$ 0.67)    |
| Avg. no. rated items per user | 106.04  | 56.46<br>( $\pm$ 30.56)      | 84.84<br>( $\pm$ 0.00)   | 28.09<br>( $\pm$ 9.43)    |
| Max. no. rated items per user | 737     | 450.80<br>( $\pm$ 213.68)    | 650.20<br>( $\pm$ 35.37) | 251.40<br>( $\pm$ 45.59)  |
| Min. no. rated items per user | 20      | 3.70<br>( $\pm$ 3.16)        | 6.40<br>( $\pm$ 2.07)    | 1<br>( $\pm$ 0.00)        |
| Max. no. users rating an item | 583     | 293.30<br>( $\pm$ 182.82)    | 466.40<br>( $\pm$ 14.06) | 120.20<br>( $\pm$ 9.71)   |
| Min. no. users rating an item | 1       | 1<br>( $\pm$ 0.00)           | 1<br>( $\pm$ 0.00)       | 1<br>( $\pm$ 0.00)        |

**Table A.1. Summary of statistics of the MovieLens 100K dataset.**

| Property                      | Overall   | Average                           | Average in training         | Average in test             |
|-------------------------------|-----------|-----------------------------------|-----------------------------|-----------------------------|
| No. users                     | 6040      | 6038.40<br>( $\pm$ 1.84)          | 6040<br>( $\pm$ 0.00)       | 6036.80<br>( $\pm$ 1.10)    |
| No. items                     | 3706      | 3,576.50<br>( $\pm$ 109.48)       | 3,680.20<br>( $\pm$ 6.26)   | 3,472.80<br>( $\pm$ 6.61)   |
| No. ratings                   | 1,000,000 | 500,104.50<br>( $\pm$ 316,293.86) | 800,167.20<br>( $\pm$ 0.45) | 200,041.80<br>( $\pm$ 0.45) |
| Density                       | 4.47%     | 2.28%<br>( $\pm$ 1.39)            | 3.60<br>( $\pm$ 0.01)       | 0.95<br>( $\pm$ 0.00)       |
| Avg. no. rated items per user | 165.60    | 82.81<br>( $\pm$ 52.36)           | 132.48<br>( $\pm$ 0.00)     | 33.14<br>( $\pm$ 0.01)      |
| Max. no. rated items per user | 2,314     | 1,157<br>( $\pm$ 732.10)          | 1,851.20<br>( $\pm$ 24.00)  | 462.80<br>( $\pm$ 24.00)    |
| Min. no. rated items per user | 20        | 5.90<br>( $\pm$ 5.22)             | 10.80<br>( $\pm$ 1.10)      | 1<br>( $\pm$ 0.00)          |
| Max. no. users rating an item | 3,428     | 1,714<br>( $\pm$ 1,084.24)        | 2,742.40<br>( $\pm$ 22.95)  | 685.60<br>( $\pm$ 22.95)    |
| Min. no. users rating an item | 1         | 1<br>( $\pm$ 0.00)                | 1<br>( $\pm$ 0.00)          | 1<br>( $\pm$ 0.00)          |

**Table A.2. Summary of statistics of the MovieLens 1M dataset.**

Table A.1 and Table A.2 show some statistics about the MovieLens datasets. It is interesting to note that the overall sparsity in both datasets is similar (between 4% and 6%), but the number of users vs. items is quite different: in MovieLens 100K there are more items than users, whereas this situation in MovieLens 1M is the opposite.

Finally, in those experiments where we evaluated content-based recommenders, we used extended versions of the MovieLens dataset with information extracted from the Internet Movie Database<sup>9</sup>, such as the movies' directors, actors, and genres. Details about how this information was gathered and merged with the MovieLens user profiles can be found in (Cantador, 2008).

### A.1.2 Last.fm dataset

Among the two Last.fm datasets distributed by Ò. Celma (Celma and Herrera, 2008), and described in Section 3.4.2, in this thesis we used the one denoted as Last.fm 1K, since it contains the music listening history (scrobbles) of each user at the track level, and includes the timestamp at which a user listened to a track.

For our experiments we built two versions of that dataset. Table A.3 shows some statistics about them. For building the first version (referred as Last.fm dataset from now on), we applied a 5-fold cross-validation on 80-20% training-test data splits. For building the second version, we performed a single temporal splitting of the user scrobbles, maintaining an 80-20% training-test ratio. This is equivalent to divide the data at some timestamp in such a way that 80% of the scrobbles are con-

---

<sup>9</sup> The Internet Movie Database, IMDb, <http://www.imdb.com>

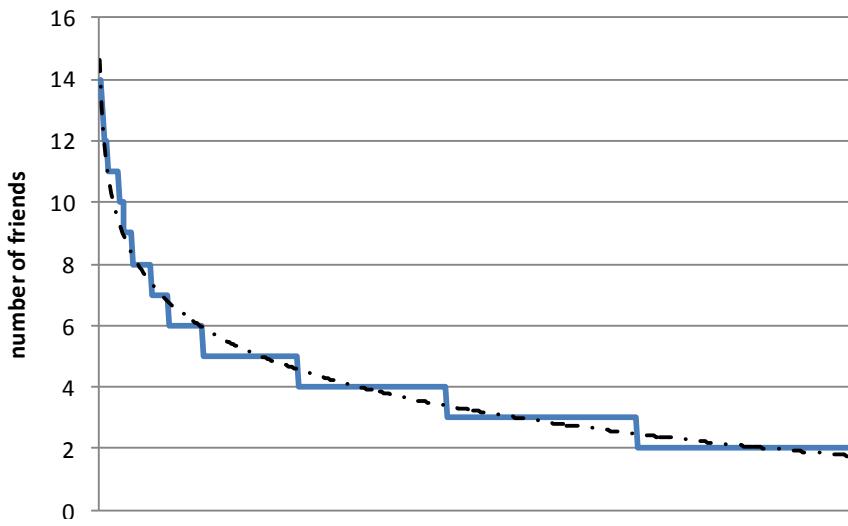
| Property                          | Overall         | Five-fold                                    |   |  | Temporal                                     |                 |                |
|-----------------------------------|-----------------|--|---|--|--|-----------------|----------------|
|                                   |                 | Average                                      | Average<br>in training                        | Average<br>in test                           | Average                                      | Training        | Test           |
| No. users                         | 992             | 991<br>( $\pm 1.70$ )                        | 992<br>( $\pm 0.00$ )                         | 990<br>( $\pm 2.00$ )                        | 932<br>( $\pm 16.97$ )                       | 920             | 944            |
| No. items                         | 176,892         | 108,065.20<br>( $\pm 48,266.14$ )            | 153,854.20<br>( $\pm 155.01$ )                | 62,276.20<br>( $\pm 199.80$ )                | 118,530<br>( $\pm 43,371.10$ )               | 149,198         | 87,862         |
| No. scrobbblings                  | 19,129,595      | 9,564,797.50<br>( $\pm 6,049,542.74$ )       | 15,303,676<br>( $\pm 56,393.95$ )             | 3,825,919<br>( $\pm 56,393.95$ )             | 9,564,798.50<br>( $\pm 8,115,999.81$ )       | 15,303,677      | 3,825,920      |
| No. unique scrobblings            | 904,309         | 452,154.50<br>( $\pm 285,967.77$ )           | 723,447.20<br>( $\pm 313.46$ )                | 180,861.80<br>( $\pm 313.46$ )               | 539,553<br>( $\pm 268,287.63$ )              | 729,261         | 349,845        |
| Density                           | 10.90%<br>0.52% | 8.12% ( $\pm 2.02$ )<br>0.38% ( $\pm 0.10$ ) | 10.03% ( $\pm 0.04$ )<br>0.47% ( $\pm 0.00$ ) | 6.21% ( $\pm 0.11$ )<br>0.29% ( $\pm 0.00$ ) | 7.88% ( $\pm 4.62$ )<br>0.48% ( $\pm 0.08$ ) | 11.15%<br>0.53% | 4.61%<br>0.42% |
| Avg. no. scrobbblings per user    | 19,283.87       | 9,645.83<br>( $\pm 6,094.22$ )               | 15,427.09<br>( $\pm 56.85$ )                  | 3,864.57<br>( $\pm 57.35$ )                  | 10,343.66<br>( $\pm 8,896.50$ )              | 16,634.43       | 4,052.88       |
| Max.no. scrobbblings per user     | 183,094         | 93,795.20<br>( $\pm 55,859.24$ )             | 146,475.20<br>( $\pm 7,036.93$ )              | 41,115.20<br>( $\pm 5,753.13$ )              | 117,872.50<br>( $\pm 84,792.71$ )            | 177,830         | 57,915         |
| Min. no. scrobbblings per user    | 2               | 1.30<br>( $\pm 0.48$ )                       | 1.60<br>( $\pm 0.55$ )                        | 1<br>( $\pm 0.00$ )                          | 1<br>( $\pm 0.00$ )                          | 1               | 1              |
| Avg. no. scrobbled items per user | 911.60          | 455.99<br>( $\pm 288.08$ )                   | 729.28<br>( $\pm 0.32$ )                      | 182.69<br>( $\pm 0.12$ )                     | 581.64<br>( $\pm 298.45$ )                   | 792.68          | 370.60         |
| Max. no. scrobbled items per user | 8,452           | 4,226.00<br>( $\pm 2,672.76$ )               | 6,761.60<br>( $\pm 0.55$ )                    | 1,690.40<br>( $\pm 0.55$ )                   | 5,707<br>( $\pm 1,554.22$ )                  | 6,806           | 4,608          |
| Min. no. scrobbled items per user | 2               | 1.30<br>( $\pm 0.48$ )                       | 1.60<br>( $\pm 0.55$ )                        | 1<br>( $\pm 0.00$ )                          | 1<br>( $\pm 0.00$ )                          | 1               | 1              |
| Max. no. users scrobbing an item  | 710             | 356.10<br>( $\pm 233.52$ )                   | 568.00<br>( $\pm 10.39$ )                     | 144.20<br>( $\pm 7.36$ )                     | 527.50<br>( $\pm 146.37$ )                   | 631             | 424            |
| Min. no. users scrobbing an item  | 1               | 1<br>( $\pm 0.00$ )                          | 1<br>( $\pm 0.00$ )                           | 1<br>( $\pm 0.00$ )                          | 1<br>( $\pm 0.00$ )                          | 1               | 1              |
| Time interval (in days)           | 3,150           | 1,586.85<br>( $\pm 0.02$ )                   | 1,586.85<br>( $\pm 0.00$ )                    | 1,586.84<br>( $\pm 0.02$ )                   | 1,574.87<br>( $\pm 331.64$ )                 | 1,340           | 1,809          |

**Table A.3. Summary of statistics of the Last.fm datasets.**

tained in the training split. In the built dataset (referred as Last.fm temporal dataset from now on), the above timestamp is 16th October, 2008.

In both datasets we aggregated the user listening data by artist (amounting to  $|J| = 176,892$  instead of  $|J| \approx 960,000$  if tracks were used) in order to overcome the sparsity at track level. Apart from that, it is also interesting to note the difference between considering separate scrobblings (when a user listens to an artist, in our setting) against considering unique scrobblings (or number of user-item pairs), which corresponds to the typical situation in movie recommendation (where, for each movie, there are not several ratings given by a particular user).

Furthermore, in some experiments we transformed log-based information to explicit ratings by using the method described in (Celma, 2010) and (Celma, 2008). This method takes into account the number of times a user listened to an artist, in such a way that the artists located in the 80-100% interquartile range of the user's listening distribution receive a rating of 5 (in a five point scale), those in the next interquartile range are mapped to a rating of 4, and so on. Additionally, the use of the coefficient of variation  $CV(u) = \sigma(u)/\mu(u)$  is proposed in (Celma, 2008) to discriminate between skewed and uniform distributions. We have not considered this coefficient since it produced strange behaviours in the recommenders, such as too many ties in the recommended items and errors in the computation of some correlations (since the mean would have the same value for every rating, see Equation (2.5)).



**Figure A.1.** Friend distribution among the users composing the original test set of the CAMRa '10 social track. Note that a logarithmic regression line fits almost perfectly with the above distribution. The total number of users is 439, and the maximum and minimum numbers of friends are 14 and 2 respectively.

On this dataset, content-based recommenders used the artists' tags. The considered tags for each artist were the most popular tags assigned to that artist according to the Last.fm API<sup>10</sup>.

### A.1.3 CAMRa dataset

Among the several datasets provided in the different CAMRa challenges (Said et al., 2010) (Said et al., 2011), in this thesis we used the dataset published at the social track of CAMRa '10. This dataset was gathered from the Filmtipset community, and contains social links between users, movie ratings, movie comments, and other attributes of users and movies. The design of this dataset and, more specifically, the selection of test users as provided in the challenge's social track is representative of online applications in which every target user has a non-empty list of contacts (see Figure A.1). This is the case of social-centric systems such as Facebook, LinkedIn, and Twitter, but not of many social media applications, such as Delicious and Last.fm, where the coverage of their social network is partial – not all registered users really use the social part of the system.

In fact, the Filmtipset dataset belongs to the latter case. Considering the set of all users, more than 10,000 out of about 16,500 users do not have any friends in the system. The number of contacts per user follows a power law distribution, where the average number of friends per user is 0.95, and the mode among users (with at least one friend) is 1. If we take this dataset as representative of social media systems, the

<sup>10</sup> Documentation related with the method used, <http://www.lastfm.es/api/show/artist.getTopTags>

| <b>Property</b>               | <b>Overall</b> | <b>Social</b>   |             | <b>Collaborative-Social</b> |             |
|-------------------------------|----------------|-----------------|-------------|-----------------------------|-------------|
|                               |                | <b>Training</b> | <b>Test</b> | <b>Training</b>             | <b>Test</b> |
| No. users                     | 16,473         | 16,473          | 439         | 16,473                      | 793         |
| No. items                     | 24,222         | 24,222          | 1,915       | 24,212                      | 2670        |
| No. ratings                   | 3,091,075      | 3,075,346       | 15,729      | 3,069,888                   | 21,187      |
| Density                       | 0.77%          | 0.77%           | 1.87%       | 0.77%                       | 1.00%       |
| Avg. no. rated items per user | 187.64         | 186.69          | 35.83       | 186.36                      | 26.72       |
| Max. no. rated items per user | 3,435          | 3,435           | 314         | 3,435                       | 314         |
| Min. no. rated items per user | 1              | 1               | 1           | 1                           | 1           |
| Max. no. users rating an item | 14,339         | 14,290          | 111         | 14,255                      | 134         |
| Min. no. users rating an item | 1              | 1               | 1           | 1                           | 1           |
| Avg no. friends per user      | 0.95           | 0.95            | 3.85        | 0.95                        | 2.13        |
| Max. no. friends per user     | 24             | 24              | 14          | 24                          | 14          |
| Min. no. friends per user     | 0              | 0               | 2           | 0                           | 0           |

**Table A.4. Summary of statistics of the CAMRa datasets.**

presence of contacts by itself does not guarantee the accuracy of social recommendation. Moreover, intermediate cases, where social data is available but not enough to support optimal recommendation, would rather seem to be the norm. We therefore simulate an alternative scenario by adding an equal amount of users without friends to a new test set by sampling randomly the same number of test users in the original test set (i.e., 439 users), but forcing them to have no friends. We name the original dataset as CAMRa Social, and the modified one as CAMRa Collaborative-Social or simply CAMRa Collaborative, since it is not focused on social information. Table A.4 shows some statistics about these datasets, from which the minimum number of friends per user in the test sets is the main difference between the above datasets (2 in the original, social dataset, and 0 in the collaborative test).

## A.2 Configuration of recommendation algorithms

In this section we provide details about the implementation of the recommendation algorithms used in Chapters 4, 6, 7, and 8. Table A.5 shows a summary of such recommenders, along with references to the chapters where the recommenders were evaluated. In the following we describe each of these recommenders, and provide their parameter values, explicit formulations, and references. The recommenders, categorised according to the type of recommendations they provide, are the following:

| Recommender | Chapter(s) where the recommender is evaluated |
|-------------|---|
| CB          | Chapters 6 and 7                              |
| IB          | Chapters 6 and 7                              |
| ItemPop     | Chapters 4, 6, and 7                          |
| kNN         | Chapters 4, 6, and 7                          |
| MF          | Chapter 4                                     |
| pLSA        | Chapters 4, 6, and 7                          |
| Random      | Chapters 4 and 6                              |
| Resnick     | Chapter 8                                     |
| TFL1        | Chapters 6 and 7                              |
| TFL2        | Chapters 6 and 7                              |

**Table A.5.** List of the recommenders evaluated in this thesis, and the chapters where their evaluations are reported.

### Non-personalised recommenders

- **ItemPop:** a non-personalised recommender based on the popularity of the item being recommended. It basically counts the number of training ratings of an item, and generates a recommendation score for the item based on such number.
- **Random:** a non-personalised recommender that generates a random recommendation score in a required value range.

### Content-based recommender

- **CB:** a content-based recommender, which, similarly to the one described in (Martinez et al., 2009), computes the cosine similarity between user and item vectors whose components can represent any possible content-based feature. A different configuration of features is used depending on the dataset. For the MovieLens dataset, we used item attributes such as movie genre, director, and country, as appeared in IMDb. Other features like actors and keywords were also tested, but yielded worse performance. Specifically, we used the most popular 3 countries for each item, 3 directors, and 8 genres per movie, as suggested in (Cantador, 2008); besides, each of these features was weighted as follows: the country feature was assigned a weight of 0.26, director, 0.06, and genre, 0.66. For the Last.fm dataset, we used as features the 50 most popular tags related to each artist.

### Rating-based recommenders

- **IB:** an item-based collaborative filtering recommender, in which Equation (2.8) is used along with Pearson's correlation as the similarity metric between items

(analogous to Equation (2.5), but considering items instead of users). Moreover, a constraint was implemented in order to remove some noise from neighbour items: only predictions produced by at least two similar items were considered; that is, if we replaced  $S_i$  by  $\mathcal{I}_u$ , then those users with only one item similar to the target item  $i$  in their profiles did not receive any recommendation.

- **kNN:** a user-based (nearest neighbour) collaborative filtering recommender, in which a slight modification of Equation (2.3) is used to adapt the item-based algorithm proposed in (Koren, 2008). We used 100 neighbours and Pearson's correlation as similarity metric, as defined in Equation (2.5). Specifically, we considered the following equation:

$$g(u, i) = b(u, i) + \frac{1}{\sum_{v \in N_k(u, i)} sim_{sh}(u, v)} \sum_{v \in N_k(u, i)} sim_{sh}(u, v)(r(v, i) - b(v, i))$$

where  $sim_{sh}(u, v)$  is the shrunk similarity  $sim(u, v)n(u, v)/(n(u, v) + \lambda_2)$ ,  $n(u, v)$  being  $|\mathcal{I}_u \cap \mathcal{I}_v|$ , that is, the number of users who rated both items. Besides,  $b(u, i) = \mu + b_u + b_i$  is the user-item bias learnt solving the following least squares problem, where  $\mu$  indicates the overall average rating:

$$\min_{b_u, b_i} \sum_{(u, i)} \left[ (r(u, i) - \mu - b_u - b_i)^2 + \lambda_1 \left( \sum_u b_u^2 + \sum_i b_i^2 \right) \right]$$

In our experiments no tuning of the  $\lambda_1$  regularisation parameter was done (partially because this method indeed optimises RMSE, which is not our main goal), and used a fixed value of 0.02. We used a learning rate of 0.005, and 100 iterations in the optimisation process. Furthermore, we did not shrink the similarity, so an effective  $\lambda_2 = 0$  was used. Additionally, at least 5 neighbours had to participate in the prediction process to consider a predicted item as valid.

- **MF:** a matrix factorisation recommender, as implemented in the Mahout library by means of the Expectation Maximisation (EM) algorithm. We used 100 iterations and 50 features, leaving the rest of the parameters as default (i.e., 0.005 as learning rate, 0.02 as regularisation parameter, and 0.005 as random noise). We have to note that the original method proposed in (Koren et al., 2009) used Alternating Least Squares to learn the user and item factorised vectors. However, the version implemented in Mahout provided worse results for this learning method, and thus we used the EM algorithm in our experiments.
- **pLSA:** a probabilistic Latent Semantic Analysis recommender, in which we use the co-occurrence latent semantic model as defined in (Hofmann, 2004). That

is, the prediction is made by computing  $p(i|u) = \sum_z p(i|z)p(z|u)$ . The models based on ratings produced worse results, and are not applicable with implicit (log-based) user preference data. This is why we preferred to use the co-occurrence model over the one based on ratings. In the experiments we used 50 factors and 50 iterations.

- **Resnick:** a user-based (nearest neighbour) collaborative filtering recommender, implemented as in Equation (2.3), where rating deviations from the user's and neighbour's rating means are considered (Resnick et al., 1994). As discussed in Chapter 8, different neighbourhood sizes were tested. We used Pearson's correlation as the user similarity metric, like in Equation (2.5).
- **TFL1:** an item-based collaborative filtering recommender, which uses the TF method with normalisation  $s_{00}$ , as defined in (Bellogín et al., 2011b). This is equivalent to an standard item-based CF algorithm without dividing by the similarities values, that is:

$$g(u, i) = \sum_{j \in S_i} sim(i, j)r(u, j)$$

We did not consider the neighbourhood size, replacing  $S_i$  by  $\mathcal{I}_u$ , and using Pearson's correlation as similarity metric  $sim(i, j)$ .

- **TFL2:** an item-based collaborative filtering recommender in which, similarly to the TFL1 recommender, we used the TF method with normalisation  $s_{01}$ , and  $L_2$  norm as defined in (Bellogín et al., 2011b). This is equivalent to an standard item-based CF, but instead of normalising by the sum of similarity values, it divides by the square root of the sum of similarity values squared, that is:

$$g(u, i) = \frac{1}{\sqrt{\sum_{j \in S_i} sim(i, j)^2}} \sum_{j \in S_i} sim(i, j)r(u, j)$$

Like before, we did not consider the neighbourhood size, and used Pearson's correlation as the similarity metric.

## Social-based recommenders

- **Personal:** a social filtering recommender presented in (Ben-Shimon et al., 2007), which utilises Equation (2.11) for score prediction. We set  $K = 2$  and  $L = 6$ , along with a constraint specifying that at least two friends have to participate on the item prediction in order to be considered valid.
- **PureSocial:** a social filtering recommender, which is an adaptation of the standard user-based collaborative filtering in which friends are used as neighbours,

as proposed in (Liu and Lee, 2010) and (Bellogín et al., 2012). Specifically, we used Equation (2.4) as the user-based method, but where the neighbourhood  $N_k(u, i)$  is built by means of the user's social network. No neighbourhood size is used, and, like in the previous recommender, at least two friends have to participate on the item prediction to consider the suggestion as a valid one.

We implemented these recommenders on top of the Mahout library<sup>11</sup>, and will make the developed source code publicly available at the following URL: <http://ir.ii.uam.es/~alejandro/thesis>.

Additionally, we implemented the static and dynamic weighted recommender ensembles evaluated in Chapter 7 using the rank fusion library provided in (Fernández et al., 2006a) and (Fernández et al., 2006b). This library contains a series of techniques for the two basic stages of any rank fusion problem: score normalisation and score combination. In this thesis we conducted the following procedure: a) taking each item ranking (in a user basis) for every recommender in an ensemble, b) normalising each ranking using either rank or score normalisation techniques (Renda and Straccia, 2003), c) combining the normalised rankings using a weighted sum to compute the score of each item (i.e., combSUM method); the weight assigned to each ranking may come from a performance predictor, as explained in Section 7.2.3, and d) ranking the items according to the scores produced in the last combination stage.

### A.3 Configuration of evaluation methodologies

In Chapter 4 we presented several evaluation methodologies that have been further elaborated and used in the experiments of Chapters 6 and 7. In this section we provide specific examples regarding how these methodologies build the set of items to recommend. We also indicate the value of the parameters required by each method that have been used in the different chapters of this dissertation.

In Chapter 4 we classified the different target item selection strategies according to three design settings: the base candidate settings (AI or TI), the relevant item selection (AR or 1R), and the non-relevant item selection (AN or NN). Table A.6 shows the specific configurations of these methodologies, according to the notation introduced in Chapter 4. Note that the uniform methodologies (U1R, UAR, and uuUAR) take as additional parameters  $\eta$  and  $\xi$ ; for the MovieLens 1M dataset, these parameters took the following values:  $\eta = 118$ ,  $v = 350$ , and  $\xi = 99$ .

---

<sup>11</sup> Available at <http://mahout.apache.org>

| Methodology | Base candidate | Relevant item selection | Non-relevant item selection | Configuration  | Chapters    |
|-------------|----------------|-------------------------|-----------------------------|--|-------------|
| 1R          | TI             | 1R                      | NN                          | $N_u = 99$   | 4, 6, and 7 |
| AR          | TI             | AR                      | AN                          |  | 4, 6, and 7 |
| P1R         | TI             | 1R                      | NN                          | $N_u = 99$<br>$m = 10$   | 4, 6, and 7 |
| U1R         | TI             | 1R                      | NN                          | $N_u = 99$<br>$r_{test}(i) = \eta, \forall i$                                  | 4, 6, and 7 |
| UAR         | AI/TI          | AR                      | AN                          | $r_{test}(i) = \eta, \forall i$  | 4           |
| uuU1R       | TI             | 1R                      | NN                          | $N_u = 99$<br>$r_{test}(i) = \nu, \forall i$<br>$r_{test}(u) = \xi, \forall u$ | 6 and 7     |

**Table A.6. Configuration of the evaluation methodologies used in the thesis.**

In the following we present several toy examples to illustrate how the different methodologies behave. For these examples we consider three users as follows:

- A user  $u_1$  with training set  $R_{train}(u_1) = \{i_1, i_2\}$ , and test set  $R_{test}(u_1) = \{i_3, i_4\}$ .
- A user  $u_2$  with  $R_{train}(u_2) = \{i_1, i_2, i_3\}$ , and test set  $R_{test}(u_2) = \{i_5\}$ .
- A user  $u_3$  with  $R_{train}(u_3) = \{i_3, i_4, i_5\}$ , and test set  $R_{test}(u_3) = \{i_1, i_2\}$ .

According to these training and test data, we next compare the target items selected for user  $u_1$  by 1R and AR methodologies. The AR methodology selects all items contained in the test set (TI-AN) as non-relevant, and all items in the active user's test (AR); hence, it produces the following set to be scored and ranked by a given recommender:  $\{i_3, i_4, i_5\}$ . The 1R methodology, on the other hand, produces a ranking for each relevant item as follows:  $\{i_3, i_5\}$  for item  $i_3$ , and  $\{i_4, i_5\}$  for item  $i_4$ , where we use  $N_u = 1$  for illustration purposes.

In the example we also have that  $r(u_1) = r(u_2) = 4$ , and  $r(u_3) = 5$ , and for the items  $r(i_1) = r(i_2) = r(i_3) = 3$  and  $r(i_4) = r(i_5) = 2$ , where  $r(u)$  denotes the number of items rated by a user (and similarly for items) as denoted in Chapter 4. With the uniform methodologies we have to build a different training/test split for each user. Specifically, in the U1R and UAR methodologies we need to ensure that every item has been rated the same number of times in the test, whereas in the uuU1R methodology we also have the constraint that all users have to appear the same number of times in the test set. Therefore, the following split would be valid for the U1R or UAR methodologies:

$$\begin{aligned} R_{train}(u_1) &= \{i_4\}; R_{test}(u_1) = \{i_1, i_2, i_3\} \\ R_{train}(u_2) &= \{i_2, i_3, i_5\}; R_{test}(u_2) = \{i_1\} \\ R_{train}(u_3) &= \{i_1, i_4, i_5\}; R_{test}(u_3) = \{i_2, i_3\} \end{aligned}$$

In this case  $\eta = 2$  for every item in the test set. Then, we can apply the 1R or AR methodology to obtain the corresponding rankings for the U1R or UAR meth-

odologies, respectively. However, since we have a different amount of ratings for each user, this configuration is not valid for the uuU1R methodology. A valid configuration for this methodology would be:

$$\begin{aligned} R_{train}(u_1) &= \{i_3, i_4\}; R_{test}(u_1) = \{i_1, i_2\} \\ R_{train}(u_2) &= \{i_2, i_5\}; R_{test}(u_2) = \{i_1, i_3\} \\ R_{train}(u_3) &= \{i_1, i_4, i_5\}; R_{test}(u_3) = \{i_2, i_3\} \end{aligned}$$

Here, we have  $\nu = 2$  and  $\xi = 2$  for all the users and items in the test set. In this context, if we apply the 1R methodology to this split we would obtain results corresponding to the uuU1R methodology.

Alternatively, for uuU1R we can also derive a valid configuration directly from a given uniform split, i.e., the one used for U1R and UAR. To do this, we would exploit the degree of freedom we have to select the set of not-relevant items for each user. Hence, we have to guarantee that all the non-relevant items selected for each user have to appear the same number of times in every target set to be recommended. This constraint would be satisfied depending on the user and item distributions, where a greedy algorithm (by brute force) could be applied if enough ratings are available. For instance, in our previous example we cannot derive a valid configuration from the split presented for U1R and UAR. In such a case, an alternative split should be made as explained above.

For simplicity purposes, in the examples we have assumed that all items contained in the test set are relevant. However, this is not the general case, and a threshold on the minimum rating value should be set. To be consistent across methodologies, only items rated as 5 by a user are considered relevant. Thus, for the AR methodology, although every item in a user's test set is considered, in the evaluation only those items with 5 stars are considered relevant. On the other hand, the 1R methodology builds one ranking for each relevant item. That is, in the example above, if  $r(u_1, i_3) = 5$  and  $r(u_1, i_4) = 3$ , then a unique ranking would be generated and evaluated, the one corresponding to  $\{i_3, i_5\}$ .

As additional material, we will make publicly available an implementation of the above evaluation methodologies at <http://ir.ii.uam.es/~alejandro/thesis>.

## A.4 Additional results about correlations

Next we report additional experiments to those presented in Chapter 6 regarding the computation of correlation coefficients between the performance predictors and recommenders. Specifically, we provide experimental results obtained with metrics different to P@10, and alternative correlation coefficients such as Spearman's and Kendall's.

### A.4.1 Correlations of user predictors using rating data

In this section we provide results obtained by using different correlation coefficients and metrics in the evaluation of user predictors based on ratings. Specifically, we provide results obtained with Spearman's  $\rho$  and Kendall's  $\tau$  correlation coefficients, and metrics such as MAP@10 (because it is considered as more stable than precision (Manning et al., 2008; Baeza-Yates and Ribeiro-Neto, 2011)), recall@10 (as an inversely related metric to precision), and nDCG@50 (because it includes graded relevance and a different cutoff).

Table A.7, Table A.8, and Table A.9 show the results obtained when the above three metrics are used along with the Pearson's correlation coefficient and the AR methodology. Comparing these results with those shown in Table 6.7, we can observe that most of the correlation trends are similar to the ones presented in Chapter 6. Specifically, the results with nDCG@50 are almost equivalent to those obtained with P@10, proving that our predictors are also consistent with other metrics apart from precision. On the other hand, the correlation values with MAP@10 and recall@10 metrics have some differences with respect to the P@10 metric, being lower in general. In fact, the correlation with CB and pLSA recommenders is negative, probably due to the (inverse) relation between precision and recall, which makes very difficult to optimise both metrics at the same time.

| Predictor          | Random | CB     | IB     | ItemPop | kNN    | pLSA   | TFL1  | TFL2   |
|--------------------|--------|--------|--------|---------|--------|--------|-------|--------|
| Count (training)   | 0.005  | -0.031 | 0.009  | 0.047   | 0.094  | -0.049 | 0.024 | 0.281  |
| Count (test)       | 0.004  | -0.029 | 0.009  | 0.047   | 0.092  | -0.052 | 0.022 | 0.276  |
| Mean               | 0.009  | 0.010  | -0.002 | -0.065  | -0.036 | -0.002 | 0.007 | -0.103 |
| Standard deviation | 0.004  | 0.013  | 0.011  | -0.018  | -0.029 | -0.023 | 0.015 | -0.069 |
| ItemSimple Clarity | 0.007  | -0.031 | 0.010  | 0.040   | 0.089  | -0.054 | 0.026 | 0.274  |
| ItemUser Clarity   | 0.005  | -0.029 | 0.011  | 0.039   | 0.089  | -0.054 | 0.028 | 0.272  |
| RatUser Clarity    | 0.006  | -0.031 | 0.009  | 0.048   | 0.093  | -0.049 | 0.024 | 0.273  |
| RatItem Clarity    | 0.005  | -0.029 | 0.008  | 0.041   | 0.087  | -0.053 | 0.022 | 0.267  |
| IRUser Clarity     | 0.006  | -0.026 | 0.005  | 0.051   | 0.083  | -0.046 | 0.021 | 0.265  |
| IRItem Clarity     | 0.003  | -0.021 | 0.009  | 0.038   | 0.076  | -0.046 | 0.023 | 0.234  |
| IRUserItem Clarity | 0.005  | -0.025 | 0.007  | 0.049   | 0.082  | -0.047 | 0.024 | 0.261  |
| Entropy            | 0.000  | -0.032 | 0.007  | 0.040   | 0.103  | -0.037 | 0.021 | 0.296  |

**Table A.7. Pearson's correlation values between rating-based user predictors and MAP@10 for different recommenders, using the AR methodology on the MovieLens 1M dataset.**

| Predictor          | Random | CB     | IB     | ItemPop | kNN    | pLSA   | TFL1  | TFL2   |
|--------------------|--------|--------|--------|---------|--------|--------|-------|--------|
| Count (training)   | -0.001 | -0.048 | 0.009  | 0.012   | 0.053  | -0.120 | 0.020 | 0.213  |
| Count (test)       | -0.002 | -0.047 | 0.008  | 0.009   | 0.051  | -0.123 | 0.018 | 0.208  |
| Mean               | 0.009  | 0.009  | -0.002 | -0.067  | -0.024 | -0.017 | 0.005 | -0.104 |
| Standard deviation | 0.004  | 0.014  | 0.011  | -0.038  | -0.034 | -0.035 | 0.019 | -0.076 |
| ItemSimple Clarity | 0.000  | -0.049 | 0.009  | 0.000   | 0.047  | -0.131 | 0.023 | 0.203  |
| ItemUser Clarity   | -0.001 | -0.046 | 0.011  | 0.003   | 0.049  | -0.121 | 0.025 | 0.205  |
| RatUser Clarity    | 0.000  | -0.049 | 0.009  | 0.011   | 0.055  | -0.116 | 0.020 | 0.201  |
| RatItem Clarity    | -0.002 | -0.046 | 0.007  | 0.008   | 0.050  | -0.115 | 0.019 | 0.199  |
| IRUser Clarity     | 0.001  | -0.040 | 0.004  | 0.018   | 0.047  | -0.108 | 0.018 | 0.201  |
| IRItem Clarity     | -0.002 | -0.036 | 0.006  | 0.006   | 0.039  | -0.106 | 0.020 | 0.176  |
| IRUserItem Clarity | 0.000  | -0.040 | 0.006  | 0.016   | 0.046  | -0.109 | 0.021 | 0.197  |
| Entropy            | -0.004 | -0.048 | 0.006  | 0.010   | 0.058  | -0.124 | 0.018 | 0.252  |

**Table A.8. Pearson's correlation values between rating-based user predictors and recall@10 for different recommenders, on the MovieLens 1M dataset and using the AR methodology**

| Predictor          | Random | CB    | IB     | ItemPop | kNN    | pLSA   | TFL1  | TFL2   |
|--------------------|--------|-------|--------|---------|--------|--------|-------|--------|
| Count (training)   | 0.085  | 0.012 | 0.010  | 0.221   | 0.228  | 0.144  | 0.152 | 0.528  |
| Count (test)       | 0.085  | 0.015 | 0.010  | 0.225   | 0.231  | 0.148  | 0.153 | 0.525  |
| Mean               | 0.023  | 0.037 | -0.014 | -0.035  | 0.011  | 0.050  | 0.040 | -0.084 |
| Standard deviation | 0.003  | 0.019 | 0.010  | -0.046  | -0.069 | -0.048 | 0.021 | -0.096 |
| ItemSimple Clarity | 0.094  | 0.020 | 0.011  | 0.226   | 0.235  | 0.156  | 0.173 | 0.540  |
| ItemUser Clarity   | 0.084  | 0.014 | 0.013  | 0.205   | 0.220  | 0.134  | 0.167 | 0.513  |
| RatUser Clarity    | 0.087  | 0.005 | 0.010  | 0.221   | 0.240  | 0.139  | 0.160 | 0.517  |
| RatItem Clarity    | 0.081  | 0.008 | 0.008  | 0.201   | 0.218  | 0.123  | 0.158 | 0.493  |
| IRUser Clarity     | 0.079  | 0.022 | -0.001 | 0.216   | 0.201  | 0.136  | 0.138 | 0.492  |
| IRItem Clarity     | 0.074  | 0.032 | 0.007  | 0.184   | 0.173  | 0.119  | 0.145 | 0.440  |
| IRUserItem Clarity | 0.079  | 0.023 | 0.003  | 0.212   | 0.198  | 0.133  | 0.146 | 0.485  |
| Entropy            | 0.078  | 0.025 | 0.004  | 0.224   | 0.235  | 0.192  | 0.127 | 0.561  |

**Table A.9. Pearson's correlation values between rating-based predictors and nDCG@50 for different recommenders, on the MovieLens 1M dataset and using the AR methodology.**

As pointed out by other authors, Pearson's correlation values by themselves may not be enough to completely understand the relation between analysed variables (Hauff et al., 2009; Carmel and Yom-Tov, 2010). Because of that, in Table A.10 and Table A.11 we provide results found when Spearman's and Kendall's correlations are used, along with the P@10 metric and the AR methodology. Comparing these results with those shown in Table 6.7, we can observe that strong correlations are also obtained using non parametric coefficients such as Kendall's  $\tau$ . More specifically, our results show that  $r \geq \rho \geq \tau$ , with respect to the Pearson's  $r$ , Spearman's  $\rho$ , and Kendall's  $\tau$  correlation coefficients.

| Predictor          | Random | CB    | IB    | ItemPop | kNN    | pLSA   | TFL1  | TFL2   |
|--------------------|--------|-------|-------|---------|--------|--------|-------|--------|
| Count (training)   | 0.112  | 0.165 | 0.020 | 0.457   | 0.367  | 0.465  | 0.124 | 0.585  |
| Count (test)       | 0.114  | 0.174 | 0.020 | 0.473   | 0.375  | 0.484  | 0.124 | 0.589  |
| Mean               | 0.015  | 0.064 | 0.000 | 0.010   | 0.025  | 0.106  | 0.030 | -0.024 |
| Standard deviation | 0.007  | 0.005 | 0.009 | -0.032  | -0.038 | -0.037 | 0.009 | -0.064 |
| ItemSimple Clarity | 0.117  | 0.176 | 0.021 | 0.474   | 0.382  | 0.492  | 0.130 | 0.602  |
| ItemUser Clarity   | 0.112  | 0.168 | 0.021 | 0.442   | 0.362  | 0.457  | 0.131 | 0.583  |
| RatUser Clarity    | 0.113  | 0.157 | 0.021 | 0.469   | 0.388  | 0.482  | 0.122 | 0.598  |
| RatItem Clarity    | 0.113  | 0.169 | 0.019 | 0.460   | 0.378  | 0.477  | 0.130 | 0.592  |
| IRUser Clarity     | 0.110  | 0.164 | 0.019 | 0.449   | 0.364  | 0.454  | 0.123 | 0.571  |
| IRItem Clarity     | 0.107  | 0.174 | 0.017 | 0.422   | 0.318  | 0.414  | 0.123 | 0.532  |
| IRUserItem Clarity | 0.110  | 0.164 | 0.020 | 0.447   | 0.362  | 0.453  | 0.125 | 0.571  |
| Entropy            | 0.112  | 0.166 | 0.020 | 0.460   | 0.369  | 0.468  | 0.124 | 0.588  |

**Table A.10.** Spearman's correlation values between rating-based user predictors and P@10 for different recommenders, on the MovieLens 1M dataset and using the AR methodology.

| Predictor          | Random | CB    | IB    | ItemPop | kNN    | pLSA   | TFL1  | TFL2   |
|--------------------|--------|-------|-------|---------|--------|--------|-------|--------|
| Count (training)   | 0.092  | 0.134 | 0.017 | 0.358   | 0.296  | 0.353  | 0.102 | 0.471  |
| Count (test)       | 0.094  | 0.143 | 0.017 | 0.373   | 0.305  | 0.371  | 0.102 | 0.477  |
| Mean               | 0.013  | 0.052 | 0.000 | 0.008   | 0.020  | 0.079  | 0.025 | -0.018 |
| Standard deviation | 0.006  | 0.004 | 0.008 | -0.024  | -0.030 | -0.028 | 0.007 | -0.050 |
| ItemSimple Clarity | 0.096  | 0.143 | 0.017 | 0.371   | 0.308  | 0.374  | 0.106 | 0.484  |
| ItemUser Clarity   | 0.091  | 0.136 | 0.018 | 0.344   | 0.292  | 0.346  | 0.107 | 0.467  |
| RatUser Clarity    | 0.093  | 0.127 | 0.017 | 0.367   | 0.313  | 0.366  | 0.100 | 0.481  |
| RatItem Clarity    | 0.092  | 0.137 | 0.016 | 0.359   | 0.304  | 0.362  | 0.106 | 0.476  |
| IRUser Clarity     | 0.090  | 0.133 | 0.015 | 0.350   | 0.293  | 0.344  | 0.100 | 0.457  |
| IRItem Clarity     | 0.087  | 0.141 | 0.014 | 0.328   | 0.255  | 0.312  | 0.101 | 0.423  |
| IRUserItem Clarity | 0.090  | 0.133 | 0.017 | 0.348   | 0.292  | 0.343  | 0.102 | 0.456  |
| Entropy            | 0.092  | 0.134 | 0.017 | 0.359   | 0.297  | 0.355  | 0.101 | 0.472  |

**Table A.11.** Kendall's correlation values between rating-based user predictors and P@10 for different recommenders, on the MovieLens 1M dataset and using the AR methodology.

| Predictor          | Random | CB     | IB     | kNN    | TFL1   | TFL2   |
|--------------------|--------|--------|--------|--------|--------|--------|
| Count (training)   | 0.288  | 0.255  | 0.170  | 0.488  | 0.529  | 0.545  |
| Count (test)       | 0.384  | 0.384  | 0.242  | 0.592  | 0.602  | 0.623  |
| Mean               | -0.063 | -0.009 | -0.060 | -0.018 | -0.103 | 0.012  |
| Standard deviation | -0.011 | -0.033 | 0.045  | -0.016 | 0.031  | -0.135 |
| ItemSimple Clarity | 0.282  | 0.257  | 0.146  | 0.491  | 0.521  | 0.564  |
| ItemUser Clarity   | 0.289  | 0.255  | 0.189  | 0.479  | 0.540  | 0.530  |
| RatUser Clarity    | 0.282  | 0.234  | 0.182  | 0.469  | 0.507  | 0.516  |
| RatItem Clarity    | 0.239  | 0.191  | 0.184  | 0.395  | 0.442  | 0.426  |
| IRUser Clarity     | 0.149  | 0.171  | -0.092 | 0.257  | 0.253  | 0.399  |
| IRItem Clarity     | 0.232  | 0.218  | 0.152  | 0.372  | 0.453  | 0.416  |
| IRUserItem Clarity | 0.279  | 0.265  | 0.105  | 0.444  | 0.523  | 0.545  |
| Entropy            | 0.263  | 0.256  | 0.110  | 0.497  | 0.499  | 0.574  |

**Table A.12.** Pearson's correlation values between rating-based user predictors and nDCG@50 for different recommenders, on the MovieLens 100K dataset and using the AR methodology.

We also compare the results shown in Table A.9 with those obtained on different datasets. Table A.12 shows the correlation values for the MovieLens 100K dataset, as published in (Bellogín et al., 2011b). We observe that the correlation does not

| Predictor          | Random | CB     | IB     | ItemPop | kNN    | pLSA   | TFL1   | TFL2   |
|--------------------|--------|--------|--------|---------|--------|--------|--------|--------|
| Count (training)   | 0.389  | 0.464  | 0.100  | 0.718   | 0.553  | 0.697  | 0.490  | 0.793  |
| Count (test)       | 0.392  | 0.475  | 0.100  | 0.728   | 0.562  | 0.711  | 0.492  | 0.797  |
| Mean               | -0.093 | -0.151 | -0.022 | -0.117  | -0.041 | -0.152 | -0.123 | -0.104 |
| Standard deviation | 0.017  | 0.036  | 0.009  | -0.029  | -0.045 | -0.025 | 0.018  | -0.069 |
| ItemSimple Clarity | 0.383  | 0.453  | 0.100  | 0.721   | 0.563  | 0.700  | 0.478  | 0.802  |
| ItemUser Clarity   | 0.391  | 0.465  | 0.112  | 0.696   | 0.544  | 0.678  | 0.514  | 0.783  |
| RatUser Clarity    | 0.387  | 0.442  | 0.116  | 0.702   | 0.557  | 0.665  | 0.498  | 0.788  |
| RatItem Clarity    | 0.366  | 0.426  | 0.096  | 0.663   | 0.524  | 0.633  | 0.500  | 0.765  |
| IRUser Clarity     | 0.376  | 0.469  | 0.068  | 0.671   | 0.498  | 0.664  | 0.499  | 0.742  |
| IRItem Clarity     | 0.358  | 0.443  | 0.082  | 0.622   | 0.469  | 0.609  | 0.482  | 0.684  |
| IRUserItem Clarity | 0.378  | 0.470  | 0.083  | 0.664   | 0.494  | 0.658  | 0.513  | 0.736  |
| Entropy            | 0.313  | 0.442  | 0.056  | 0.687   | 0.508  | 0.747  | 0.341  | 0.710  |

**Table A.13.** Pearson’s correlation values between rating-based user predictors and P@10 for different recommenders, on the MovieLens 1M dataset and using the AR methodology, but considering all the items in test set as relevant.

change significantly; only the correlation values for the CB, IB and TFL1 recommenders increase in the MovieLens 100K dataset.

Additionally, as shown in Chapter 6, when different experimental designs are tested, the selection of relevant and not relevant items is very important. In the AR methodology, as described in Section A.3, we consider as relevant those items whose ratings are 5 (to some extent in order to be consistent with the rest of the methodologies). Table A.13, on the other hand, shows that the correlation changes when all the items in the test set are considered relevant. The first thing to note when we compare these results with those shown in Table 6.7 is that the absolute correlation values are now much higher. However, if we take the correlations with the Random recommender as a reference, these relative correlations are very similar in both tables. Moreover, the trend in predictive power of the predictors (that is, which predictors have more or less predictive power) is consistent across this dimension, which evidences the stability of the evaluation methodology used to measure the predictive power of the recommendation performance predictors.

#### A.4.2 Correlations of user predictors using log data

In this section we focus on complementing our results with correlations between the predictor values and the MAP@10 metric, since we have observed in the previous sections that other correlation coefficients are consistent with Pearson’s.

Hence, in Table A.14 we report the results with the temporal split, whereas in Table A.15 we present the results for the random split on the Last.fm dataset. Respectively, we have to compare these tables against Table 6.14 and Table 6.15. This comparison shows that the results obtained using the precision and MAP metrics are equivalent, in contrast to what happened in the previous sections. This is because in this experiment we use the 1R methodology where there is only one relevant item, in

| Predictor             | Random | CB     | ItemPop | kNN    | pLSA   |
|-----------------------|--------|--------|---------|--------|--------|
| Average Count         | 0.001  | 0.173  | 0.027   | -0.054 | 0.163  |
| Count                 | 0.028  | 0.188  | -0.044  | 0.140  | 0.115  |
| Mean                  | -0.059 | -0.407 | 0.037   | -0.089 | -0.220 |
| Standard deviation    | -0.034 | -0.191 | -0.035  | -0.119 | -0.094 |
| Autocorrelation       | 0.049  | 0.156  | -0.079  | -0.105 | 0.001  |
| TimeSimple Clarity    | -0.044 | -0.435 | 0.053   | -0.257 | -0.212 |
| ItemTime Clarity      | 0.024  | 0.142  | 0.085   | 0.316  | 0.084  |
| ItemPriorTime Clarity | 0.069  | 0.219  | 0.240   | 0.345  | 0.204  |
| Frequency Clarity     | -0.039 | -0.421 | -0.248  | -0.307 | -0.365 |
| ItemSimple Clarity    | 0.008  | 0.118  | -0.052  | 0.275  | 0.014  |

**Table A.14. Pearson's correlation values between log-based user predictors and MAP@10 for different recommenders, on the Last.fm temporal dataset and using the 1R methodology.**

| Predictor             | Random | CB     | ItemPop | kNN    | pLSA   |
|-----------------------|--------|--------|---------|--------|--------|
| Average Count         | -0.043 | -0.065 | -0.139  | 0.020  | -0.136 |
| Count                 | -0.031 | -0.212 | -0.194  | -0.040 | -0.260 |
| Mean                  | 0.004  | 0.173  | 0.111   | 0.008  | 0.117  |
| Standard deviation    | -0.010 | 0.116  | 0.116   | 0.021  | 0.099  |
| Autocorrelation       | 0.010  | -0.032 | -0.078  | -0.007 | -0.063 |
| TimeSimple Clarity    | 0.041  | 0.304  | 0.277   | 0.082  | 0.344  |
| ItemTime Clarity      | 0.037  | -0.147 | 0.020   | 0.052  | -0.084 |
| ItemPriorTime Clarity | 0.036  | -0.028 | 0.210   | 0.149  | 0.072  |
| Frequency Clarity     | 0.001  | -0.038 | -0.286  | -0.158 | -0.211 |
| ItemSimple Clarity    | 0.028  | -0.240 | -0.131  | -0.021 | -0.213 |

**Table A.15. Pearson's correlation values between log-based user predictors and MAP@10 for different recommenders, on the Last.fm five-fold dataset and using the 1R methodology.**

contrast to the AR methodology used in that experiment, and thus, these two metrics are equivalent. Similar results are obtained for recall and nDCG metrics for the same reasons.

### A.4.3 Correlations of user predictors using social data

In this section we present additional results regarding the correlations between social-based performance predictors and evaluation metrics. Like in the previous section, here we only show correlations with respect to the MAP@10 metric, which, in this case, do not provide results equal to those of the precision metric since we use the AR methodology instead of the 1R methodology.

Table A.16 shows Pearson's correlation values on the social version of the CAMRa dataset. We observe that the correlations are much lower than those presented in Table 6.16; in some situations even the sign of the correlation changes, like for most of the values of kNN. A more interesting situation is observed on the CAMRa Collaborative dataset (Table A.17), where strong but negative correlations arise, in particular for the ItemPop and pLSA recommenders. This result may have a direct impact on the performance of the dynamic ensembles, since the correlation of

| Predictor              | Random | ItemPop | kNN    | pLSA   | Personal | PureSocial |
|------------------------|--------|---------|--------|--------|----------|------------|
| Count (training)       | 0.016  | 0.104   | 0.022  | 0.024  | 0.013    | 0.066      |
| Count (test)           | 0.086  | -0.032  | 0.021  | -0.167 | -0.215   | -0.294     |
| Mean                   | -0.047 | 0.074   | 0.075  | 0.009  | 0.003    | 0.025      |
| Standard deviation     | -0.030 | -0.065  | -0.144 | -0.061 | -0.063   | -0.004     |
| Avg neighbour degree   | -0.025 | -0.057  | -0.031 | -0.124 | -0.120   | -0.240     |
| Betweenness centrality | -0.015 | 0.061   | -0.010 | -0.015 | 0.011    | -0.076     |
| Clustering coefficient | 0.028  | -0.024  | 0.035  | -0.106 | 0.001    | -0.133     |
| Degree                 | -0.026 | -0.070  | -0.065 | -0.130 | -0.184   | -0.185     |
| Ego components size    | -0.044 | 0.019   | -0.051 | 0.029  | -0.036   | 0.021      |
| HITS                   | -0.010 | -0.018  | 0.056  | -0.002 | 0.082    | 0.040      |
| PageRank               | -0.020 | -0.022  | -0.047 | 0.041  | -0.059   | 0.041      |
| Two-hop neighbourhood  | -0.039 | -0.040  | -0.062 | -0.118 | -0.148   | -0.227     |
| ItemSimple Clarity     | 0.012  | 0.132   | 0.031  | 0.037  | 0.023    | 0.080      |

**Table A.16.** Pearson's correlation values between social-based user predictors and MAP@10 for different recommenders, on the CAMRa social dataset and using the AR methodology.

| Predictor              | Random | ItemPop | kNN    | pLSA   | Personal | PureSocial |
|------------------------|--------|---------|--------|--------|----------|------------|
| Count (training)       | 0.004  | -0.072  | 0.054  | -0.096 | 0.012    | 0.067      |
| Count (test)           | 0.031  | -0.119  | 0.056  | -0.163 | -0.215   | -0.294     |
| Mean                   | -0.045 | 0.119   | 0.042  | 0.057  | 0.005    | 0.026      |
| Standard deviation     | 0.041  | -0.043  | -0.139 | -0.041 | -0.063   | -0.004     |
| Avg neighbour degree   | 0.086  | -0.168  | 0.044  | -0.148 | -0.120   | -0.240     |
| Betweenness centrality | -0.005 | 0.010   | -0.008 | -0.039 | 0.012    | -0.075     |
| Clustering coefficient | -0.002 | -0.133  | 0.099  | -0.149 | -0.001   | -0.135     |
| Degree                 | 0.045  | -0.174  | 0.039  | -0.198 | -0.183   | -0.185     |
| Ego components size    | 0.030  | -0.126  | 0.016  | -0.143 | -0.035   | 0.022      |
| HITS                   | -0.009 | -0.023  | 0.027  | 0.085  | 0.082    | 0.040      |
| PageRank               | 0.000  | -0.136  | 0.024  | -0.134 | -0.058   | 0.041      |
| Two-hop neighbourhood  | 0.080  | -0.148  | 0.016  | -0.151 | -0.146   | -0.227     |
| ItemSimple Clarity     | 0.002  | -0.068  | 0.057  | -0.092 | 0.022    | 0.081      |

**Table A.17.** Pearson's correlation values between social-based user predictors and MAP@10 for different recommenders, on the CAMRa collaborative dataset and using the AR methodology.

the predictors is now very different. We refer the reader to Section A.5.3 where we show how the dynamic ensembles perform when this metric is used.

## A.5 Additional results about dynamic ensembles

Next we present additional results obtained in the experiments aimed to compare static and dynamic hybrid recommendations. We report values of metrics different to P@10, which has been extensively used in the thesis. In particular, we focus on MAP@10 in order to provide a full overview of the predictors' behaviour, since correlations with respect to this metric have been presented in the previous sections.

### A.5.1 Performance results from dynamic ensembles on rating data

In this section we report experiments where dynamic hybrid recommenders are built by means of rating-based performance predictors. Table A.18 and Table A.19 show performance values (in terms of the MAP metric) of the hybrid recommenders by using the AR and 1R methodologies respectively. Additionally, Table A.20 shows performance values using item predictors along with the uuU1R methodology.

We can observe that the results from Table A.18 are quite similar to those presented in Table 7.2: in most cases the dynamic hybrid recommenders outperform the baseline static recommender, and the best result for each ensemble is obtained either by using the perfect correlation predictor or one of the clarity-based performance predictors. The only difference of these results with those of the P@10 metric (Table 7.2) is that when we evaluate with MAP@10 the baseline outperforms the dynamic ensembles for the combination HRU3. We have to note that the best static recommender is very different for this combination: whereas for P@10 the best result is obtained when  $\lambda = 0.9$ , for MAP@10 the best result is obtained for  $\lambda = 0.5$ , where, as we observed in Chapter 7, the rating-based user predictors seem to perform worse when the best static recommender is close to that value.

|                                  | HRU1           | HRU2           | HRU3          | HRU4           | HRU5           | HRU6           |
|----------------------------------|----------------|----------------|---------------|----------------|----------------|----------------|
| R1 ( $\lambda=1.0$ )             | 0.0005         | 0.0298         | 0.0116        | 0.0116         | 0.0116         | 0.0116         |
| R2 ( $\lambda=0.0$ )             | 0.0086         | 0.0086         | 0.0086        | 0.0001         | 0.1047         | 0.0551         |
| Baseline ( $\lambda=0.5$ )       | 0.0043         | 0.0268         | 0.0271        | 0.0003         | 0.0794         | 0.0499         |
| Best static<br>(best $\lambda$ ) | 0.0087         | 0.0310         | 0.0271        | 0.0022         | 0.1108         | 0.0584         |
| Perfect correlation              | <u>0.0099</u>  | 0.0373         | 0.0297        | <u>0.0119</u>  | 0.1166         | <u>0.0663</u>  |
| PC-OM                            | 0.0097         | <u>0.0399</u>  | <u>0.0305</u> | 0.0045         | 0.1125         | 0.0602         |
| PC-FW                            | 0.0097         | 0.0296         | 0.0278        | 0.0008         | 0.1136         | 0.0615         |
| Entropy-OM                       | <b>0.0057▼</b> | <b>0.0333△</b> | 0.0260▼       | <b>0.0009▼</b> | <b>0.0863▼</b> | <b>0.0509△</b> |
| ItemSimple-OM                    | <b>0.0090△</b> | <b>0.0330△</b> | 0.0256▼       | <b>0.0009▼</b> | <b>0.1149△</b> | <b>0.0572▼</b> |
| ItemUser-OM                      | <b>0.0091△</b> | <b>0.0332△</b> | 0.0255▼       | <b>0.0008▼</b> | <b>0.1161△</b> | <b>0.0576▼</b> |
| RatUser-OM                       | <b>0.0093△</b> | <b>0.0335△</b> | 0.0262▼       | <b>0.0009▼</b> | <b>0.1178△</b> | <b>0.0575▼</b> |
| RatItem-OM                       | <b>0.0093△</b> | <b>0.0329△</b> | 0.0259▼       | <b>0.0008▼</b> | <b>0.1185△</b> | <b>0.0576▼</b> |
| IRUser-OM                        | <b>0.0087△</b> | <b>0.0326△</b> | 0.0257▼       | <b>0.0008▼</b> | <b>0.1146△</b> | <b>0.0580▼</b> |
| IRItem-OM                        | <b>0.0091△</b> | <b>0.0321△</b> | 0.0250▼       | <b>0.0008▼</b> | <b>0.1160△</b> | <b>0.0577▼</b> |
| IRUserItem-OM                    | <b>0.0090△</b> | <b>0.0325△</b> | 0.0256▼       | <b>0.0008▼</b> | <b>0.1154△</b> | <b>0.0578▼</b> |
| Entropy-FW                       | <b>0.0054▼</b> | <b>0.0282▼</b> | 0.0264▼       | <b>0.0006▼</b> | <b>0.0849▼</b> | <b>0.0508△</b> |
| ItemSimple-FW                    | <b>0.0085▼</b> | <b>0.0281▼</b> | 0.0263▼       | <b>0.0006▼</b> | <b>0.1111△</b> | <b>0.0573▼</b> |
| ItemUser-FW                      | <b>0.0088△</b> | <b>0.0282▼</b> | 0.0262▼       | <b>0.0006▼</b> | <b>0.1131△</b> | <b>0.0571▼</b> |
| RatUser-FW                       | <b>0.0090△</b> | <b>0.0281▼</b> | 0.0265▼       | <b>0.0006▼</b> | <b>0.1147△</b> | <b>0.0569▼</b> |
| RatItem-FW                       | <b>0.0089△</b> | <b>0.0278▼</b> | 0.0264▼       | <b>0.0006▼</b> | <b>0.1154△</b> | <b>0.0573▼</b> |
| IRUser-FW                        | <b>0.0085▼</b> | <b>0.0280▼</b> | 0.0266▼       | <b>0.0007▼</b> | <b>0.1094▼</b> | <b>0.0568▼</b> |
| IRItem-FW                        | <b>0.0089△</b> | <b>0.0278▼</b> | 0.0256▼       | <b>0.0006▼</b> | <b>0.1123△</b> | <b>0.0570▼</b> |
| IRUserItem-FW                    | <b>0.0085▼</b> | <b>0.0281▼</b> | 0.0265▼       | <b>0.0007▼</b> | <b>0.1116△</b> | <b>0.0573▼</b> |

**Table A.18. Dynamic ensemble performance values (MAP@10) using the rating-based user predictors, on the MovieLens 1M and using the AR methodology.**

Table A.19 shows performance values of the hybrid recommenders using the 1R methodology. The outcome of this experiment is identical to that presented in Table 7.3, except that now the best performing ensemble is a dynamic hybrid recommenders, either the perfect correlation or the PC-OM, instead of the best static ensemble, further validating our framework.

Additionally, in Table A.20 we show the performance of the dynamic hybrid recommenders using item predictors with the MAP metric. We may observe that these results are very similar to those presented for P@10 in Table 7.9, which emphasises the flexibility of our approach, in terms of being able to obtain performance improvements when using different evaluation metrics.

|                                  | HRU1            | HRU2            | HRU3            | HRU4            | HRU5            | HRU6            |
|----------------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| R1 ( $\lambda=1.0$ )             | 0.0559          | 0.3335          | 0.1815          | 0.1815          | 0.1815          | 0.1815          |
| R2 ( $\lambda=0.0$ )             | 0.0847          | 0.0847          | 0.0847          | 0.0125          | 0.5163          | 0.3430          |
| Baseline ( $\lambda=0.5$ )       | 0.1147          | 0.2384          | 0.1974          | 0.0989          | 0.4004          | 0.3211          |
| Best static<br>(best $\lambda$ ) | 0.1186<br>(0.3) | 0.3369<br>(0.9) | 0.2248<br>(0.8) | 0.1789<br>(0.9) | 0.5189<br>(0.1) | 0.3618<br>(0.1) |
| Perfect correlation              | <u>0.1409</u>   | <u>0.3625</u>   | <u>0.2584</u>   | <u>0.1919</u>   | 0.5139          | <u>0.3822</u>   |
| PC-OM                            | 0.1176          | 0.3014          | 0.2380          | 0.1375          | <u>0.5213</u>   | 0.3785          |
| PC-FW                            | 0.1155          | 0.2678          | 0.2141          | 0.1189          | 0.5012          | 0.3715          |
| Entropy-OM                       | 0.1138▼         | <b>0.3187▼</b>  | <b>0.2103▼</b>  | <b>0.1383▼</b>  | 0.3802▼         | 0.3087▼         |
| ItemSimple-OM                    | 0.1107▼         | <b>0.3210▼</b>  | <b>0.2111▼</b>  | <b>0.1398▼</b>  | <b>0.5022▼</b>  | <b>0.3517▼</b>  |
| ItemUser-OM                      | 0.1084▼         | <b>0.3174▼</b>  | <b>0.2097▼</b>  | <b>0.1379▼</b>  | <b>0.5093▼</b>  | <b>0.3534▼</b>  |
| RatUser-OM                       | 0.1051▼         | <b>0.3216▼</b>  | <b>0.2130▼</b>  | <b>0.1405▼</b>  | <b>0.5162▼</b>  | <b>0.3556▼</b>  |
| RatItem-OM                       | 0.1046▼         | <b>0.3187▼</b>  | <b>0.2120▼</b>  | <b>0.1400▼</b>  | <b>0.5168▼</b>  | <b>0.3560▼</b>  |
| IRUser-OM                        | 0.1109▼         | <b>0.3131▼</b>  | <b>0.2092▼</b>  | <b>0.1382▼</b>  | <b>0.4991▼</b>  | <b>0.3495▼</b>  |
| IRItem-OM                        | 0.1073▼         | <b>0.3062▼</b>  | <b>0.2025▼</b>  | <b>0.1328▼</b>  | <b>0.5030▼</b>  | <b>0.3484▼</b>  |
| IRUserItem-OM                    | 0.1082▼         | <b>0.3127▼</b>  | <b>0.2091▼</b>  | <b>0.1381▼</b>  | <b>0.5035▼</b>  | <b>0.3497▼</b>  |
| Entropy-FW                       | <b>0.1160▼</b>  | <b>0.2703▼</b>  | <b>0.2042▼</b>  | <b>0.1184▼</b>  | 0.3965▼         | 0.3196▼         |
| ItemSimple-FW                    | 0.1142▼         | <b>0.2712▼</b>  | <b>0.2049▼</b>  | <b>0.1192▼</b>  | <b>0.4864▼</b>  | <b>0.3491▼</b>  |
| ItemUser-FW                      | 0.1120▼         | <b>0.2700▼</b>  | <b>0.2037▼</b>  | <b>0.1181▼</b>  | <b>0.4951▼</b>  | <b>0.3509▼</b>  |
| RatUser-FW                       | 0.1097▼         | <b>0.2713▼</b>  | <b>0.2058▼</b>  | <b>0.1194▼</b>  | <b>0.5049▼</b>  | <b>0.3535▼</b>  |
| RatItem-FW                       | 0.1095▼         | <b>0.2707▼</b>  | <b>0.2046▼</b>  | <b>0.1193▼</b>  | <b>0.5066▼</b>  | <b>0.3542▼</b>  |
| IRUser-FW                        | 0.1146▼         | <b>0.2699▼</b>  | <b>0.2044▼</b>  | <b>0.1183▼</b>  | <b>0.4828▼</b>  | <b>0.3474▼</b>  |
| IRItem-FW                        | 0.1109▼         | <b>0.2672▼</b>  | <b>0.2010▼</b>  | <b>0.1158▼</b>  | <b>0.4905▼</b>  | <b>0.3461▼</b>  |
| IRUserItem-FW                    | 0.1123▼         | <b>0.2697▼</b>  | <b>0.2043▼</b>  | <b>0.1181▼</b>  | <b>0.4894▼</b>  | <b>0.3481▼</b>  |

**Table A.19. Dynamic ensemble performance values (MAP@10) using the rating-based user predictors, on the MovieLens 1M dataset and using the 1R methodology.**

|                                  | HRI1            | HRI2            | HRI3            | HRI4            |
|----------------------------------|-----------------|-----------------|-----------------|-----------------|
| R1 ( $\lambda=1.0$ )             | <b>0.2498</b>   | <b>0.2498</b>   | 0.0835          | 0.0835          |
| R2 ( $\lambda=0.0$ )             | 0.0717          | 0.0914          | 0.0717          | 0.0914          |
| Baseline ( $\lambda=0.5$ )       | 0.1550          | 0.1746          | 0.0878          | 0.0932          |
| Best static<br>(best $\lambda$ ) | 0.2146<br>(0.9) | 0.2233<br>(0.9) | 0.0892<br>(0.7) | 0.0954<br>(0.2) |
| Entropy-OM                       | 0.1283▼         | 0.1412▼         | 0.0872▼         | <b>0.0979▲</b>  |
| UserSimple-OM                    | <b>0.2116▲</b>  | <b>0.2273▲</b>  | <b>0.0879▼</b>  | <b>0.0975▲</b>  |
| UserItem-OM                      | <b>0.2129▲</b>  | <b>0.2267▲</b>  | 0.0866▼         | <b>0.0973▲</b>  |
| RatItem-OM                       | <b>0.2166▲</b>  | <b>0.2305▲</b>  | 0.0872▼         | <b>0.0975▲</b>  |
| RatUser-OM                       | <b>0.2231▲</b>  | <b>0.2385▲</b>  | 0.0870▼         | <b>0.0977▲</b>  |
| URItem-OM                        | <b>0.2021▼</b>  | <b>0.2136▼</b>  | 0.0869▼         | <b>0.0973▲</b>  |
| URUser-OM                        | <b>0.2176▲</b>  | <b>0.2312▲</b>  | 0.0856▼         | <b>0.0974▲</b>  |
| URIItemUser-OM                   | <b>0.2071▼</b>  | <b>0.2200▼</b>  | 0.0870▼         | <b>0.0973▲</b>  |
| Entropy-FW                       | 0.1382▼         | 0.1517▼         | 0.0873▼         | <b>0.0981▲</b>  |
| UserSimple-FW                    | <b>0.1770▼</b>  | <b>0.1950▼</b>  | <b>0.0900▲</b>  | <b>0.0975▲</b>  |
| UserItem-FW                      | <b>0.1771▼</b>  | <b>0.1953▼</b>  | <b>0.0902▲</b>  | <b>0.0973▲</b>  |
| RatItem-FW                       | <b>0.1776▼</b>  | <b>0.1957▼</b>  | <b>0.0902▲</b>  | <b>0.0975▲</b>  |
| RatUser-FW                       | <b>0.1793▼</b>  | <b>0.1976▼</b>  | <b>0.0904▲</b>  | <b>0.0974▲</b>  |
| URItem-FW                        | <b>0.1737▼</b>  | <b>0.1907▼</b>  | <b>0.0893▲</b>  | <b>0.0969▲</b>  |
| URUser-FW                        | <b>0.1782▼</b>  | <b>0.1964▼</b>  | <b>0.0903▲</b>  | <b>0.0971▲</b>  |
| URIItemUser-FW                   | <b>0.1752▼</b>  | <b>0.1931▼</b>  | <b>0.0899▲</b>  | <b>0.0974▲</b>  |

**Table A.20. Dynamic ensemble performance values (MAP) using the rating-based item predictors, on the MovieLens 1M dataset and using the uuU1R methodology.**

## A.5.2 Performance results from dynamic ensembles on log data

In this section we compare the performance values of hybrid recommenders using the 1R methodology with log-based predictors, and P@10 and MAP@10 metrics. From Table A.21 and Table A.22 we can observe that the performance values are very similar to those shown in Table 7.11 and Table 7.12, respectively. This may be due to the fact that correlations for MAP@10 presented in Section A.4.2 were also analogous, since precision and MAP are almost equivalent under the 1R methodology as there is only one relevant item for each evaluated ranking. From Table A.21 we have to note, nonetheless, that the combination HL2 obtains worse performance values for the OM weighting strategy. Another difference is that the best performing ensemble for the combination HL2 now is achieved by the perfect correlation method, not by the best static recommender, as shown in Table 7.11 and Table 7.12. This shows that there would be room for improvement in the HL2 combination if we are able to define predictors with stronger correlation values. A similar situation is found for the combination HL3 in the five-fold split.

|                                  | HL1             | HL2             | HL3                    |
|----------------------------------|-----------------|-----------------|------------------------|
| R1 ( $\lambda=1.0$ )             | 0.4094          | 0.4094          | 0.7229                 |
| R2 ( $\lambda=0.0$ )             | <u>0.8255</u>   | 0.5601          | 0.4094                 |
| Baseline ( $\lambda=0.5$ )       | 0.6922          | 0.6244          | 0.6429                 |
| Best static<br>(best $\lambda$ ) | 0.7913<br>(0.1) | 0.6326<br>(0.3) | <u>0.7256</u><br>(0.1) |
| Perfect correlation              | 0.7485          | <u>0.6470</u>   | 0.6940                 |
| PC-OM                            | 0.7272          | 0.6239          | 0.6763                 |
| PC-FW                            | 0.7188          | 0.6240          | 0.6669                 |
| ItemSimple-OM                    | <b>0.7742</b> ▲ | 0.6235▼         | <b>0.7165</b> ▲        |
| Autocorrelation-OM               | 0.6592▼         | 0.5962▼         | 0.6163▼                |
| TimeSimple-OM                    | <b>0.7676</b> ▲ | 0.5913▼         | <b>0.6955</b> ▲        |
| ItemTime-OM                      | <b>0.7762</b> ▲ | 0.6200▼         | <b>0.7149</b> ▲        |
| ItemPriorTime-OM                 | <b>0.7354</b> ▲ | 0.6223▼         | <b>0.6777</b> ▲        |
| ItemSimple-FW                    | <b>0.7597</b> ▲ | <b>0.6329</b> ▲ | <b>0.7106</b> ▲        |
| Autocorrelation-FW               | 0.6827▼         | 0.6177▼         | 0.6353▼                |
| TimeSimple-FW                    | <b>0.7514</b> ▲ | 0.6048▼         | <b>0.6893</b> ▲        |
| ItemTime-FW                      | <b>0.7608</b> ▲ | <b>0.6292</b> ▲ | <b>0.7061</b> ▲        |
| ItemPriorTime-FW                 | <b>0.7276</b> ▲ | <b>0.6278</b> ▲ | <b>0.6714</b> ▲        |

**Table A.21. Dynamic ensemble performance values (MAP@10) using the log-based user predictors, on the Last.fm temporal split and using the 1R methodology.**

|                                  | HL1             | HL2             | HL3             |
|----------------------------------|-----------------|-----------------|-----------------|
| R1 ( $\lambda=1.0$ )             | 0.0453          | 0.0453          | 0.5824          |
| R2 ( $\lambda=0.0$ )             | 0.5901          | 0.5387          | 0.0453          |
| Baseline ( $\lambda=0.5$ )       | 0.4233          | 0.3961          | 0.3308          |
| Best static<br>(best $\lambda$ ) | 0.5728<br>(0.1) | 0.5317<br>(0.1) | 0.5463<br>(0.1) |
| Perfect correlation              | <u>0.5820</u>   | 0.5396          | 0.5728          |
| PC-OM                            | <u>0.5805</u>   | <u>0.5403</u>   | <u>0.5768</u>   |
| PC-FW                            | 0.5795          | 0.5357          | 0.5611          |
| ItemSimple-OM                    | <b>0.5395</b> ▼ | <b>0.4894</b> ▼ | <b>0.4397</b> ▲ |
| Autocorrelation-OM               | 0.3972▼         | 0.3659▼         | <b>0.3642</b> ▲ |
| TimeSimple-OM                    | <b>0.5561</b> ▲ | <b>0.5206</b> ▲ | 0.2405▼         |
| ItemTime-OM                      | <b>0.5407</b> ▲ | <b>0.4881</b> ▲ | <b>0.4375</b> ▲ |
| ItemPriorTime-OM                 | <b>0.4577</b> ▲ | <b>0.4108</b> ▲ | <b>0.4276</b> ▲ |
| ItemSimple-FW                    | <b>0.5240</b> ▲ | <b>0.4791</b> ▼ | <b>0.3826</b> ▲ |
| Autocorrelation-FW               | 0.4191▼         | 0.3896▼         | <b>0.3523</b> ▲ |
| TimeSimple-FW                    | <b>0.5372</b> ▲ | <b>0.5033</b> ▲ | 0.2813▼         |
| ItemTime-FW                      | <b>0.5243</b> ▲ | <b>0.4779</b> ▼ | <b>0.3819</b> ▲ |
| ItemPriorTime-FW                 | <b>0.4582</b> ▲ | <b>0.4201</b> ▲ | 0.3783▼         |

**Table A.22. Dynamic ensemble performance values (MAP@10) using the log-based user predictors, on the Last.fm five-fold split and using the 1R methodology.**

### A.5.3 Performance results from dynamic ensembles on social data

In this section we extend the results presented in Section 7.3.3, where P@10 and methodology AR were used in the two versions of CAMRa dataset. Here we use the same methodology, but the MAP@10 metric, as in the previous sections. In Table A.23 we can observe that there are some differences in the social version of the dataset when compared against the results shown in Table 7.14. We may attribute such differences to the different optimal static hybrid recommenders obtained since the correlations have not changed significantly. For instance, for HS1, the best lambda for static hybrids was 0.3 with P@10 and now it is 0.7 with MAP@10.

It is worth noting that even when the best static is so different from one metric to the other, the best performance is still obtained with the perfect correlation ensemble, which again reinforces the idea that finding predictors with higher correlations would be able to dynamically select the best weights in a user basis.

Finally, in Table A.24 we present the results regarding the collaborative version of the CAMRa dataset. In this case, we have a strong evidence towards the benefits of using performance predictors. First, we have to recall the absolute values of the pLSA correlations are stronger for the MAP metric than for precision. Then, now we

|                                  | HS1                                     | HS2                                     | HS3                                     | HS4                    |
|----------------------------------|---|---|---|------------------------|
| R1 ( $\lambda=1.0$ )             | 0.2002                                  | 0.2002                                  | 0.2192                                  | 0.2192                 |
| R2 ( $\lambda=0.0$ )             | 0.1203                                  | 0.0364                                  | 0.1203                                  | 0.0364                 |
| Baseline ( $\lambda=0.5$ )       | 0.2175                                  | 0.2287                                  | 0.2555                                  | 0.2398                 |
| Best static<br>(best $\lambda$ ) | 0.2222<br>(0.3)                         | 0.2349<br>(0.9)                         | 0.2630<br>(0.3)                         | 0.2408<br>(0.9)        |
| Perfect correlation              | <u>0.2632</u>                           | 0.2562                                  | 0.2652                                  | <u>0.2511</u>          |
| PC-OM                            | 0.2451                                  | <u>0.2576</u>                           | <u>0.2668</u>                           | 0.2497                 |
| PC-FW                            | 0.2355                                  | 0.2378                                  | 0.2661                                  | 0.2446                 |
| AvgNeighDeg-OM                   | <b>0.2176<math>\triangleleft</math></b> | <b>0.2403<math>\triangleleft</math></b> | <b>0.2570<math>\triangleleft</math></b> | 0.2229 $\triangledown$ |
| BetCentrality-OM                 | 0.2031 $\triangledown$                  | 0.2232 $\triangleleft$                  | 0.2146 $\triangledown$                  | 0.2236 $\triangledown$ |
| ClustCoeff-OM                    | 0.2082 $\triangledown$                  | 0.2201 $\triangledown$                  | 0.2261 $\triangledown$                  | 0.2155 $\triangledown$ |
| Degree-OM                        | 0.2147 $\triangledown$                  | <b>0.2303<math>\triangledown</math></b> | <b>0.2615<math>\triangleleft</math></b> | 0.2166 $\triangledown$ |
| EgoCompSize-OM                   | 0.2078 $\triangledown$                  | 0.2276 $\triangleleft$                  | <b>0.2577<math>\triangleleft</math></b> | 0.2214 $\triangledown$ |
| HITS-OM                          | 0.2127 $\triangledown$                  | <b>0.2428<math>\triangleleft</math></b> | 0.2187 $\triangledown$                  | 0.2231 $\triangledown$ |
| PageRank-OM                      | 0.2108 $\triangledown$                  | <b>0.2289<math>\triangleleft</math></b> | <b>0.2573<math>\triangleleft</math></b> | 0.2238 $\triangledown$ |
| TwoHopNeigh-OM                   | 0.2121 $\triangledown$                  | <b>0.2377<math>\triangleleft</math></b> | 0.2545 $\triangledown$                  | 0.2202 $\triangledown$ |
| AvgNeighDeg-FW                   | <b>0.2218<math>\triangleleft</math></b> | <b>0.2370<math>\triangleleft</math></b> | <b>0.2574<math>\triangleleft</math></b> | 0.2300 $\triangledown$ |
| BetCentrality-FW                 | 0.2171 $\triangledown$                  | <b>0.2351<math>\triangleleft</math></b> | 0.2460 $\triangledown$                  | 0.2344 $\triangledown$ |
| ClustCoeff-FW                    | 0.2129 $\triangledown$                  | <b>0.2348<math>\triangleleft</math></b> | 0.2434 $\triangledown$                  | 0.2344 $\triangledown$ |
| Degree-FW                        | <b>0.2176<math>\triangleleft</math></b> | <b>0.2373<math>\triangleleft</math></b> | <b>0.2627<math>\triangleleft</math></b> | 0.2332 $\triangledown$ |
| EgoCompSize-FW                   | 0.2124 $\triangledown$                  | <b>0.2373<math>\triangleleft</math></b> | <b>0.2616<math>\triangleleft</math></b> | 0.2352 $\triangledown$ |
| HITS-FW                          | 0.2164 $\triangledown$                  | <b>0.2380<math>\triangleleft</math></b> | 0.2405 $\triangledown$                  | 0.2331 $\triangledown$ |
| PageRank-FW                      | 0.2169 $\triangledown$                  | <b>0.2363<math>\triangleleft</math></b> | <b>0.2574<math>\triangleleft</math></b> | 0.2336 $\triangledown$ |
| TwoHopNeigh-FW                   | <b>0.2177<math>\triangleleft</math></b> | <b>0.2373<math>\triangleleft</math></b> | <b>0.2587<math>\triangleleft</math></b> | 0.2323 $\triangledown$ |

**Table A.23. Dynamic ensemble performance values (MAP@10) using the log-based user predictors, on the CAMRa social dataset and using the AR methodology.**

|                                  | HS1                                  | HS2                                  | HS3                                  | HS4                 |
|----------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|---------------------|
| R1 ( $\lambda=1.0$ )             | 0.1234                               | 0.1234                               | 0.1334                               | 0.1334              |
| R2 ( $\lambda=0.0$ )             | 0.1474                               | 0.0195                               | 0.1474                               | 0.0195              |
| Baseline ( $\lambda=0.5$ )       | 0.2190                               | 0.1441                               | 0.2359                               | 0.1520              |
| Best static<br>(best $\lambda$ ) | 0.2228<br>(0.3)                      | 0.1494<br>(0.9)                      | 0.2440<br>(0.2)                      | 0.1520<br>(0.5)     |
| Perfect correlation              | 0.2379                               | 0.1597                               | 0.2396                               | 0.1590              |
| PC-OM                            | 0.1494                               | 0.1577                               | 0.1657                               | 0.1535              |
| PC-FW                            | 0.1492                               | 0.1462                               | 0.1660                               | 0.1498              |
| AvgNeighDeg-OM                   | <b>0.2226<math>\downarrow</math></b> | <b>0.1511<math>\triangle</math></b>  | <b>0.2426<math>\downarrow</math></b> | 0.1409 $\downarrow$ |
| BetCentrality-OM                 | 0.2069 $\downarrow$                  | 0.1412 $\downarrow$                  | 0.2119 $\downarrow$                  | 0.1410 $\downarrow$ |
| ClustCoeff-OM                    | 0.2071 $\downarrow$                  | 0.1390 $\downarrow$                  | 0.2199 $\downarrow$                  | 0.1386 $\downarrow$ |
| Degree-OM                        | 0.2175 $\downarrow$                  | 0.1457 $\downarrow$                  | <b>0.2454<math>\triangle</math></b>  | 0.1378 $\downarrow$ |
| EgoCompSize-OM                   | 0.2142 $\downarrow$                  | 0.1441 $\downarrow$                  | <b>0.2417<math>\downarrow</math></b> | 0.1395 $\downarrow$ |
| HITS-OM                          | 0.2100 $\downarrow$                  | <b>0.1519<math>\triangle</math></b>  | 0.2136 $\downarrow$                  | 0.1419 $\downarrow$ |
| PageRank-OM                      | 0.2110 $\downarrow$                  | <b>0.1447<math>\downarrow</math></b> | <b>0.2416<math>\downarrow</math></b> | 0.1405 $\downarrow$ |
| TwoHopNeigh-OM                   | 0.2156 $\downarrow$                  | <b>0.1514<math>\triangle</math></b>  | <b>0.2469<math>\triangle</math></b>  | 0.1392 $\downarrow$ |
| AvgNeighDeg-FW                   | <b>0.2214<math>\downarrow</math></b> | <b>0.1501<math>\triangle</math></b>  | <b>0.2410<math>\downarrow</math></b> | 0.1464 $\downarrow$ |
| BetCentrality-FW                 | 0.2145 $\downarrow$                  | <b>0.1479<math>\downarrow</math></b> | 0.2254 $\downarrow$                  | 0.1475 $\downarrow$ |
| ClustCoeff-FW                    | 0.2167 $\downarrow$                  | <b>0.1474<math>\triangle</math></b>  | 0.2266 $\downarrow$                  | 0.1474 $\downarrow$ |
| Degree-FW                        | <b>0.2197<math>\downarrow</math></b> | <b>0.1497<math>\triangle</math></b>  | <b>0.2437<math>\downarrow</math></b> | 0.1464 $\downarrow$ |
| EgoCompSize-FW                   | 0.2189 $\downarrow$                  | <b>0.1490<math>\downarrow</math></b> | <b>0.2405<math>\downarrow</math></b> | 0.1474 $\downarrow$ |
| HITS-FW                          | 0.2157 $\downarrow$                  | <b>0.1502<math>\triangle</math></b>  | 0.2241 $\downarrow$                  | 0.1461 $\downarrow$ |
| PageRank-FW                      | 0.2170 $\downarrow$                  | <b>0.1476<math>\downarrow</math></b> | <b>0.2440<math>\triangle</math></b>  | 0.1465 $\downarrow$ |
| TwoHopNeigh-FW                   | <b>0.2216<math>\downarrow</math></b> | <b>0.1502<math>\triangle</math></b>  | <b>0.2423<math>\downarrow</math></b> | 0.1469 $\downarrow$ |

**Table A.24. Dynamic ensemble performance values (MAP@10) using the log-based user predictors, on the CAMRa collaborative dataset and using the AR methodology.**

have to note that the performance values are now better for MAP than for precision, in particular for HS1 and HS3, where dynamic hybrid recommenders outperform the baselines in a larger number of cases.

# **Appendix B**

## **Introducción**

En este capítulo presentamos una visión general de la tesis. En las Secciones B.1 y B.2 mostramos las motivaciones y objetivos de nuestro trabajo. En la Sección B.3 resumimos las principales contribuciones de la tesis, y en la Sección B.4 enumeramos las publicaciones resultantes de nuestra investigación. Finalmente, en la Sección B.5 describimos la estructura de este documento.

## B.1 Motivación

Las tecnologías de Recuperación de Información (RI) han ganado una prevalencia excepcional en las dos últimas décadas con la explosión del número de repositorios masivos de información existentes en línea, y más en particular en la *World Wide Web*. En RI se han investigado y diseñado formas que buscan maximizar el grado de satisfacción de ciertas condiciones objetivas, típicamente – aunque no necesariamente de manera única – la satisfacción del usuario. La investigación y el desarrollo en RI han girado en torno a la definición de modelos y algoritmos que mejor alcancen dicho objetivo, de metodologías y métricas que permiten evaluar cuánto de bien se consigue esta meta con diferentes sistemas, así como de teorías consolidadas que proveen una base sólida y una orientación en el desarrollo de algoritmos de RI y su consistente evaluación. Entre las muchas tendencias que han surgido desde el flujo principal de investigación y desarrollo, un nuevo reto de investigación ha empezado a ser considerado desde comienzos de los 2000: ¿es posible predecir cómo de bueno será un resultado devuelto por un sistema de RI antes de presentarlo al usuario, o incluso, antes de efectuar por completo la búsqueda del resultado por el sistema (Cronen-Townsend et al., 2002)? Esta pregunta ha dado pie a una fértil corriente de investigación en lo que se ha llamado en RI como **predicción de eficacia**.

La predicción de eficacia tiene muchos usos potenciales en RI. Desde la perspectiva del usuario proporcionaría información que puede ser usada para dirigir una búsqueda, desde la perspectiva del sistema ayudaría a distinguir consultas poco eficaces, y desde la perspectiva del administrador del sistema permitiría identificar consultas relacionadas sobre un tema específico que resultan difíciles para el motor de búsqueda. Las técnicas de predicción de eficacia se basan en el análisis y caracterización de la evidencia usada por un sistema de RI para evaluar la relevancia (utilidad, valor, etc.) de los ítems a recuperar (documentos, artículos, etc.) en tiempo de ejecución (Cronen-Townsend et al., 2002). El escenario más común en recuperación de información supone una consulta del usuario y una colección de documentos como la entrada básica para formar una lista ordenada de resultados de búsqueda, pero otros elementos adicionales pueden tenerse en cuenta para seleccionar y ordenar resultados (Baeza-Yates and Ribeiro-Neto, 2011). Cualquier información que el sistema de recuperación de información tome como entrada puede ser utilizada también para la predicción de su eficacia, y habitualmente los métodos de predicción usan información adicional. El contexto del usuario (la tarea actual, los registros de búsquedas, las preferencias, etc.), las propiedades globales de los documentos de la colección, comparaciones con respecto a (otros) elementos de referencia como datos históricos, o la salida de distintos sistemas, son algunos ejemplos de las diferentes fuentes de información de las que un predictor puede extraer evidencia.

Predecir la eficacia de un subsistema, módulo, función o entrada contrastando la estimación de eficacia de cada componente para una consulta, permite una serie de estrategias de optimización dinámicas que seleccionen en tiempo de ejecución la opción que se predice funcionará mejor o, cuando se usan sistemas a gran escala o aproximaciones híbridas, que ajusten sobre la marcha el grado de participación de cada módulo. En el campo de RI dominan los casos donde información de relevancia, sistemas de recuperación, modelos y criterios se definen en función de la fusión o combinación de sub-modelos. Sistemas personalizados de recuperación (incluyendo técnicas como búsqueda personalizada, sistemas de recomendación, filtrado colaborativo y búsqueda contextualizada) son claros ejemplos donde se puede aplicar la predicción de eficacia dado que dichos sistemas combinan varias fuentes de evidencia para la estimación de la relevancia, como pueden ser consultas explícitas, historial de búsqueda, puntuaciones de usuarios, información social, información del usuario y modelos de contexto.

La predicción de eficacia encuentra una motivación adicional en la recomendación personalizada, puesto que esas aplicaciones pueden decidir si producir recomendaciones u ocultarlas, entregando sólo las suficientemente fiables. Más aún, los Sistemas de Recomendación (SR) actuales se caracterizan por una creciente diversificación de los tipos y fuentes de datos, contenidos, evidencias y métodos disponibles para tomar decisiones y construir los resultados. En este contexto, predecir la eficacia de un método de recomendación específico o de una componente se convierte en un problema atractivo, ya que permite una combinación adecuada de las alternativas disponibles y sacar el máximo provecho de ellas, adaptando dinámicamente la estrategia de recomendación a la situación actual. La cuestión gana mayor relevancia hoy con la proliferación de técnicas de recomendación híbridas para mejorar la precisión de los métodos – siendo el premio Netflix uno de los ejemplos paradigmáticos del uso de estos métodos, donde los participantes mejor situados combinaron múltiples métodos de recomendación. Esto requiere de una investigación en aproximaciones híbridas con un nivel de mecanismos dinámicos auto-ajustables, de manera que se optimice la efectividad resultante de los sistemas de recomendación, tomando oportunidad ventaja de datos de alta calidad cuando estén disponibles, pero evitando aferrarse a estrategias fijas cuando se predice que pueden producir resultados pobres bajo ciertas condiciones.

La predicción de eficacia en RI típicamente se evalúa en términos de correlación entre el predictor y los valores de eficacia para cada consulta. Esto requiere métricas de evaluación de la eficacia fiables, las cuales han sido analizadas cuidadosamente y están actualmente bien establecidas en el área de RI, orientadas en su mayor parte a búsqueda ad-hoc. Por el contrario, la evaluación en el campo de los SR está más abierta, y la variabilidad en las técnicas de evaluación y configuraciones experimentales es significativa. Cómo medir la eficacia de un sistema de recomendación es un

asunto clave en nuestra investigación ya que las medidas de calidad del sistema pueden verse influidas por propiedades estadísticas del método de medición y/o por el diseño experimental. A lo largo de esta tesis nos centraremos en la precisión del sistema, donde hemos de evitar que si una métrica estuviera sesgada hacia algún tipo de ruido medido a la vez que la calidad de la recomendación, pues entonces un predictor que sólo capturara dicho ruido podría actuar como un predictor de eficacia erróneamente útil. Así, los sesgos estadísticos (ruidos) de las metodologías de evaluación deben ser bien entendidos para permitir una valoración significativa de los predictores de eficacia.

Tomando el estado del arte de predicción de eficacia en RI como punto de partida, el trabajo actual replantea este problema en el campo de los Sistemas de Recomendación donde ha sido escasamente considerado hasta la fecha. Investigamos definiciones adecuadas de eficacia en el contexto de los SR y los elementos a los que sensatamente se puede aplicar, analizando los sesgos estadísticos que pueden aparecer cuando se adapta el marco de evaluación de RI a SR. De este modo tomamos como dirección principal la aplicación de los predictores de eficacia para obtener mejoras en dos problemas específicos de combinación en el campo de SR, a saber, la combinación dinámica de métodos de recomendación en sistemas de recomendación híbridos, y la agregación dinámica de las señales de vecinos en filtrado colaborativo basado en usuario.

## B.2 Objetivos

El principal objetivo de la investigación presentada aquí es encontrar métodos predictivos para la eficacia de componentes específicos de sistemas de recomendación, y mejorar la eficacia de los métodos de recomendación combinados, basados en el análisis y predicción dinámicos y automáticos de la eficacia esperada de los elementos de un método compuesto, con los cuales la participación relativa de cada elemento se ajusta de acuerdo a su efectividad predicha. Para abordar estos problemas nuestro trabajo tiene los siguientes objetivos de investigación concretos:

**O1: Análisis y formalización de cómo se define y evalúa la eficacia en los sistemas de recomendación.** Dado que pretendemos predecir su eficacia necesitamos desarrollar un estudio en profundidad sobre cómo se pueden evaluar de manera fiable los sistemas de recomendación en términos de valores numéricos de una métrica. Más aún, hemos de investigar si existen sesgos en la manera en que los sistemas se evalúan – debidos tanto a metodologías de evaluación como a métricas, ya que cualquier sesgo en el proceso de evaluación podría conducir a resultados inconcluyentes o engañosos con respecto el poder predictivo de los métodos de predicción de eficacia propuestos. Si dichos sesgos existieran, intentaríamos entenderlos de manera precisa, y desarrollar metodologías que los aislaran. Además, deberíamos com-

probar la efectividad de nuestros predictores frente a la de otros métodos conocidos más básicos y observar si cambia al aislar dichos sesgos.

**O2: Adaptación y definición de técnicas de predicción de eficacia para sistemas de recomendación.** Queremos estudiar el potencial de la predicción de eficacia en problemas y escenarios específicos en el área de los Sistemas de Recomendación. Investigaremos la definición de un marco formal donde los predictores de eficacia puedan ser integrados. Como punto de partida, pretendemos explorar la adaptación de predictores específicamente efectivos en Recuperación de Información como la claridad de la consulta (Cronen-Townsend et al., 2002) al campo de la recomendación. De manera complementaria a la adaptación de técnicas conocidas, queremos investigar la definición de nuevos predictores basados en modelos de la Teoría de Información y Grafos Sociales, además de otras aproximaciones heurísticas y específicas del dominio. Una vez hayamos definido algunos predictores de eficacia para recomendación, evaluaríamos la efectividad de dichos predictores en términos de su correlación con métricas de eficacia de manera que pudiéramos estimar su poder de predicción.

**O3: Aplicación de predictores de eficacia a sistemas de recomendación compuestos e híbridos.** Queremos identificar e integrar los predictores propuestos en métodos de recomendación combinados para obtener una mejora real en la eficacia de los métodos combinados. Con este objetivo en mente consideraremos problemas donde la agregación de métodos de recomendación es necesaria, y analizaremos cómo aplicar los predictores de eficacia mencionados antes a tales problemas. Además, deberíamos realizar un estudio metodológico para la aproximación experimental, su configuración y las métricas usadas, de manera que se usen métodos base y diseños experimentales apropiados. Finalmente, evaluaremos las mejoras y beneficios de los métodos combinados cuando se utilizan los predictores de eficacia.

## B.3 Contribuciones

Esta tesis se dedica al problema de estimar la eficacia de los sistemas de recomendación para usuarios e ítems particulares. Las contribuciones particulares de esta tesis están relacionados con la evaluación de la eficacia de un sistema de recomendación y la predicción de la misma, donde hemos abordado varios problemas relacionados con ambos temas y hemos propuesto modelos y métodos novedosos, que han sido empleados en dos aplicaciones como mostraremos a continuación.

Como un primer paso, esta tesis analiza el paradigma Cranfield de evaluación de Recuperación de Información, dado que los sistemas de recomendación normalmente se consideran como un problema particular del filtrado de información, y, por tanto, de la recuperación de información en general (Belkin and Croft, 1992). En el Capítulo 4 argumentamos las diferencias involucradas en las alternativas de

**diseños experimentales a partir de las hipótesis habituales hechas en el paradigma Cranfield**, lo cual resulta en la aparición de sesgos estadísticos considerables en Sistemas de Recomendación, para los que **proponemos diferentes métodos para neutralizar estos sesgos**. De manera adicional, las siguientes contribuciones relacionadas han sido realizadas:

- Proponemos una caracterización precisa y sistemática de las alternativas para la adaptación del paradigma Cranfield a tareas de recomendación. Identificamos hipótesis y condiciones subyacentes en dicho paradigma Cranfield que no pueden ser asumidas en los experimentos habituales de recomendación.
- Detectamos y caracterizamos los sesgos estadísticos resultantes, a saber, la dispersión de test y la popularidad de los ítems, los cuales no aparecen en colecciones de test habituales en RI, pero que interfieren en experimentos de recomendación.
- Proponemos dos diseños experimentales nuevos para neutralizar estos sesgos. Observamos que una evaluación basada en percentiles reduce considerablemente el margen para el sesgo de popularidad, mientras que una aproximación basada en un test uniforme elimina cualquier ventaja estadística obtenida por tener más puntuaciones positivas de test. Más aún, encontramos que las dos propuestas discriminan bien entre recomendaciones puramente basadas en popularidad y un algoritmo de recomendación personalizado.

Además, en esta tesis **mostramos cómo las técnicas de predicción de eficacia de consultas desarrolladas en Recuperación de Información pueden ser adaptadas a los Sistemas de Recomendación, y resultar en predictores eficaces en este dominio**. Presentamos estos predictores de eficacia en el Capítulo 6, donde proponemos distintas adaptaciones del predictor de claridad de consulta basadas en distintas interpretaciones de los modelos de lenguaje subyacentes, así como con modelos de Teoría de Información y de Grafos Sociales. Más aún, en el mismo capítulo **evaluamos la efectividad de dichos predictores midiendo la correlación con respecto a métricas de eficacia**, donde también probamos los métodos propuestos en el Capítulo 4 para neutralizar los sesgos en la evaluación. A continuación resumimos las contribuciones específicas respecto a la predicción de eficacia para recomendación:

- Definimos y elaboramos varios modelos predictivos en el dominio de los sistemas de recomendación de acuerdo a diferentes formulaciones e hipótesis, y basados en tres tipos de datos de preferencia: puntuaciones, registros y sociales.
- Las formulaciones para preferencias en base a puntuaciones se basan en adaptaciones de la claridad de consulta de RI y conceptos de Teoría de la Información como la entropía. En esta adaptación proponemos distintas estimaciones

de probabilidad, donde desarrollamos derivaciones Bayesianas y estimaciones no parámetricas.

- También explotamos atributos temporales al definir los predictores basados en registros. Más específicamente, usamos una versión sensible al tiempo de la divergencia de Kullback-Leibler, junto con otros conceptos de series temporales como la autocorrelación del usuario.
- Usamos métricas basadas en Teoría de Grafos para definir predictores que aprovechan las estructuras de red social y las correlaciones entre las propiedades topológicas de los usuarios y el éxito de las recomendaciones devueltas.
- Encontramos fuertes correlaciones entre las salidas de los predictores y las métricas de eficacia, mostrando por tanto evidencia empírica del poder predictivo de las técnicas propuestas. Más aún, cuando se utilizan las metodologías no sesgadas los predictores conservan buenos valores de correlación, evidenciando que los predictores propuestos no están sólo capturando los sesgos analizados y beneficiándose de los mismos, especialmente cuando los comparamos frente a otros predictores triviales.

Finalmente, los Capítulos 7 y 8 presentan dos aplicaciones de los predictores de eficacia en Sistemas de Recomendación. En el Capítulo 7 **proponemos varios sistemas híbridos ponderados linealmente donde las ponderaciones se ajustan dinámicamente de acuerdo a la salida de los predictores**. Observamos que las correlaciones obtenidas en el Capítulo 6 ayudan a decidir cuáles son las mejores combinaciones a experimentar. Más importante aún, **la correlación entre el predictor y el algoritmo de recomendación tiende a anticipar bien cuándo un sistema híbrido mejorará con respecto a un método base**. Además, el Capítulo 8 **presenta un marco unificado donde los predictores de eficacia se usan para seleccionar y ponderar los vecinos cercanos en un algoritmo estándar de filtrado colaborativo basado en usuario**. La metodología tradicional de predicción de eficacia es adaptada y traducida a este problema, donde definimos novedosas métricas sobre la eficacia de vecinos y evaluamos el poder predictivo de los predictores.

Las contribuciones relacionadas con la parte de aplicaciones de la tesis son, en resumen, las siguientes:

- Proponemos un marco de hibridación dinámica para decidir automáticamente cuándo y cómo debería realizarse la hibridación, dependiendo de distintas condiciones, a saber: las correlaciones entre los algoritmos de recomendación y los predictores, y la eficacia relativa entre los algoritmos combinados.
- En varios experimentos con los predictores de eficacia mencionados previamente, nuestros resultados indican que una fuerte correlación con la eficacia

tiende a corresponder con mejoras en la recomendación híbrida dinámica cuando los predictores se usan para ajustar los pesos de la combinación.

- Proponemos un marco teórico para la selección y ponderación de vecinos en sistemas de recomendación basados en usuarios. Este marco se fundamenta en la predicción de eficacia estableciendo equivalencias entre la tarea de predicción de puntuaciones basada en vecindarios y una agregación dinámica de componentes, en este caso, las predicciones de cada vecino.
- Comparamos varias métricas de confianza del estado del arte así como otras técnicas para valorar vecinos, interpretadas ambas como predictores de eficacia de vecinos. También proponemos varias métricas de eficacia de vecinos que capturan diferentes nociones de la calidad de los vecinos.

## B.4 Publicaciones relacionadas con la tesis

En los siguientes artículos de revistas y conferencias internacionales presentamos descripciones, resultados y conclusiones relacionadas con esta tesis:

### Predicción de eficacia y evaluación

1. Bellogín, A., Cantador, I., Díez, F., Castells, P., and Chavarriaga, E. (2012). An empirical comparison of social, collaborative filtering, and hybrid recommenders. *ACM Transactions on Intelligent Systems and Technology*, por aparecer.
2. Bellogín, A., Castells, P., and Cantador, I. (2011). Predicting the Performance of Recommender Systems: An Information Theoretic Approach. In Amati, G. and Crestani, F., editors, *ICTIR*, volume 6931 of *Lecture Notes in Computer Science*, pages 27–39, Berlin, Heidelberg. Springer Berlin / Heidelberg.
3. Bellogín, A., Castells, P., and Cantador, I. (2011). Self-adjusting hybrid recommenders based on social network analysis. In *Proceedings of the 34<sup>th</sup> international ACM SIGIR conference on Research and development in Information*, SIGIR ’11, pages 1147–1148, New York, NY, USA. ACM.
4. Bellogín, A., Castells, P., and Cantador, I. (2011). Precision-oriented evaluation of recommender systems: an algorithmic comparison. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys ’11, pages 333–336, New York, NY, USA. ACM.
5. Bellogín, A. and Castells, P. (2010). A Performance Prediction Approach to Enhance Collaborative Filtering Performance. In Gurrin, C., He, Y., Kazai, G., Kruschwitz, U., Little, S., Roelleke, T., Rüger, S., and Rijsbergen, editors,

- Advances in Information Retrieval*, volume 5993 of *Lecture Notes in Computer Science*, pages 382–393, Berlin, Heidelberg. Springer Berlin / Heidelberg.
6. Bellogín, A. and Castells, P. (2009). Predicting neighbor goodness in collaborative filtering. In And, T. A., Yager and, R. R., And, H. B., And, H. C., and Larsen, H. L., editors, *FQAS*, volume 5822 of *Lecture Notes in Computer Science*, pages 605–616, Berlin, Heidelberg. Springer Berlin / Heidelberg.

### Recomendación basada en contenido

7. Cantador, I., Bellogín, A., and Vallet, D. (2010). Content-based recommendation in social tagging systems. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 237–240, New York, NY, USA. ACM.
8. Cantador, I., Bellogín, A., and Castells, P. (2008). News@hand: A Semantic Web Approach to Recommending News. In Nejdl, W., Kay, J., Pu, P., and Herder, E., editors, *Adaptive Hypermedia and Adaptive Web-Based Systems*, volume 5149 of *Lecture Notes in Computer Science*, chapter 34, pages 279–283. Springer Berlin / Heidelberg, Berlin, Heidelberg.
9. Cantador, I., Bellogín, A., and Castells, P. (2009). Ontology-Based Personalised and Context-Aware Recommendations of News Items. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT '08. IEEE/WIC/ACM International Conference on*, volume 1, pages 562–565.
10. Cantador, I., Bellogín, A., Fernández-Tobías, I., and López-Hernández, S. (2011a). Semantic Contextualisation of Social Tag-Based Profiles and Item Recommendations. In Huemer, C., Setzer, T., Aalst, W., Mylopoulos, J., Sadeh, N. M., Shaw, M. J., Szyperski, C., Aalst, W., Mylopoulos, J., Sadeh, N. M., Shaw, M. J., and Szyperski, C., editors, *Electronic Commerce and Web Technologies*, volume 85 of *Lecture Notes in Business Information Processing*, chapter 9, pages 101–113. Springer Berlin Heidelberg, Berlin, Heidelberg.
11. Fernández-Tobías, I., Cantador, I., and Bellogín, A. (2011). cTag: Semantic contextualisation of social tags. In *Proceedings of the Workshop on Semantic Adaptive Social Web (SASWeb 2011). CEUR Workshop Proceedings, vol. 730*, pages 45–54. RWTH, Aachen (2011).

### Recomendación basada en filtrado colaborativo

12. Bellogín, A., Wang, J., and Castells, P. Bridging Memory-Based Collaborative Filtering and Text Retrieval. *Information Retrieval Journal*, por aparecer.
13. Bellogín, A., Cantador, I., and Castells, P. A Comparative Study of Heterogeneous Item Recommendations in Social Systems. *Information Sciences*, por aparecer.

14. Bellogín, A. and Parapar, J. (2012). Using Graph Partitioning Techniques for Neighbour Selection in User-Based Collaborative Filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, RecSys '12, pages 213–216, New York, NY, USA. ACM. (premio al mejor artículo corto)
15. Bellogín, A., Wang, J., and Castells, P. (2011). Structured collaborative filtering. In *Proceedings of the 20<sup>th</sup> ACM international conference on Information and knowledge management*, CIKM '11, pages 2257–2260, New York, NY, USA. ACM.
16. Bellogín, A., Wang, J., and Castells, P. (2011). Text Retrieval Methods for Item Ranking in Collaborative Filtering. In Clough, P., Foley, C., Gurrin, C., Jones, G., Kraaij, W., Lee, H., and M Murdoch, V., editors, *Advances in Information Retrieval*, volume 6611 of *Lecture Notes in Computer Science*, chapter 30, pages 301–306. Springer Berlin / Heidelberg, Berlin, Heidelberg.
17. Bellogín, A., Cantador, I., and Castells, P. (2010). A study of heterogeneity in recommendations for a social music service. In *Proceedings of the 1<sup>st</sup> International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, HetRec '10, pages 1–8, New York, NY, USA. ACM.

### **Recomendación basada en filtrado social**

18. Díez, F., Chavarriaga, J. E., Campos, P. G., and Bellogín, A. (2010). Movie recommendations based in explicit and implicit features extracted from the Filmtipset dataset. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa '10, pages 45–52, New York, NY, USA. ACM.

### **Recomendación sensible al tiempo**

19. Campos, P. G., Bellogín, A., Díez, F., and Cantador, I. (2012). Time Feature Selection for Identifying Active Household Members. In *Proceedings of the 21<sup>st</sup> ACM international conference on Information and knowledge management*, CIKM '12, New York, NY, USA. ACM (por aparecer).
20. Campos, P. G., Díez, F., and Bellogín, A. (2011). Temporal rating habits: a valuable tool for rating discrimination. In *Proceedings of the 2<sup>nd</sup> Challenge on Context-Aware Movie Recommendation*, CAMRa '11, pages 29–35, New York, NY, USA. ACM.
21. Campos, P. G., Bellogín, A., Díez, F., and Chavarriaga, J. E. (2010). Simple time-biased KNN-based recommendations. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa '10, pages 20–23, New York, NY, USA. ACM.

## Sistemas de recomendación híbridos

22. Cantador, I., Castells, P., and Bellogín, A. (2011). An enhanced semantic layer for hybrid recommender systems. *International Journal on Semantic Web and Information Systems*, 7(1):44–78.
23. Cantador, I., Bellogín, A., and Castells, P. (2008). A multilayer ontology-based hybrid recommendation model. *AI Commun.*, 21(2-3):203–210.
24. Cantador, I., Castells, P., and Bellogín, A. (2007). Modelling Ontology-based Multilayered Communities of Interest for Hybrid Recommendations. In *Workshop on Adaptation and Personalisation in Social Systems: Groups, Teams, Communities, at the 11th International Conference on User Modeling*.

## Evaluación de la recomendación

25. Bellogín, A., Cantador, I., Castells, P., and Ortigosa, A. (2011). Discerning Relevant Model Features in a Content-based Collaborative Recommender System. In Fürnkranz, J. and Hüllermeier, E., editors, *Preference Learning*, chapter 20, pages 429–455. Springer Berlin Heidelberg, Berlin, Heidelberg.
26. Bellogín, A., Cantador, I., Castells, P., and Ortigosa, A. (2008). Discovering Relevant Preferences in a Personalised Recommender System using Machine Learning Techniques. In *Preference Learning Workshop (PL 2008), at the 8<sup>th</sup> European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD 2008)*, pages 82–96.

Estas publicaciones se relacionan con los contenidos de esta tesis como sigue. En [4] analizamos diferentes metodologías de evaluación existentes en la literatura de recomendación (Capítulos 3 y 4). En [2], [5], [6] definimos las formulaciones para el concepto de claridad de usuario basada en puntuaciones (Capítulo 6), mientras que en [1] y [3] definimos los predictores sociales (también en el Capítulo 6). Además, en [1] y [3] también investigamos el uso de los predictores de eficacia para recomendaciones híbridas dinámicas (Capítulo 7). Más aún, en [5] y [6] abordamos el problema de la ponderación de vecinos basada en los predictores de eficacia de vecinos (Capítulo 8).

Además, durante el transcurso de la tesis, la investigación presentada aquí motivó una serie de publicaciones que abordaban temas más amplios en el área, como las recomendaciones basadas en contenido [7-11], el filtrado colaborativo [12-17], las técnicas de filtrado social [18], la recomendación sensible al tiempo [19-21], los sistemas de recomendación híbrida [22-24], y la evaluación de la recomendación [25, 26]. Estas publicaciones han resultado en el uso y construcción de conjuntos de datos, el desarrollo de algoritmos y la investigación y utilización de algunas metodologías y métricas de evaluación que aparecen en esta tesis.

## Publicaciones adicionales

Trabajo preliminar relacionado con las propuestas presentadas aquí fue incluido en primer lugar en la tesis de Máster titulada “Performance prediction in recommender Systems: Application to the dynamic optimisation of aggregative methods” (Bellogín, 2009). Específicamente, en dicho trabajo proponemos el concepto de predicción de eficacia para recomendación. Además, la motivación, el potencial impacto y los principales resultados de nuestra investigación fueron publicados como contribuciones en dos simposios doctorales internacionales:

- Bellogín, A. (2011). Predicting performance in recommender systems. Doctoral Symposium. In *Proceedings of the fifth ACM conference on Recommender systems*, Rec-Sys '11, pages 371–374, New York, NY, USA. ACM.
- Bellogín, A. (2011). Performance Prediction in Recommender Systems. Doctoral Symposium. In Konstan, J., Conejo, R., Marzo, J., and Oliver, N., editors, *User Modeling, Adaptation and Personalization*, volume 6787 of *Lecture Notes in Computer Science*, pages 401–404, Berlin, Heidelberg. Springer Berlin / Heidelberg.

Además, los siguientes artículos están bajo revisión, algunos de ellos altamente relacionados con los temas de la tesis:

- Bellogín, A., Castells, P., and Cantador, I. Statistical Biases in IR Metrics for Recommender Systems: A Methodological Framework for the Adaptation of the Cranfield Paradigm. En revisión.
- Bellogín, A., Castells, P., and Cantador, I. Neighbour Selection and Weighting in User-Based Recommender Systems: A Performance Prediction Approach. En revisión.
- Parapar, J., Bellogín, A., Castells, P., and Barreiro, Á. Relevance-Based Language Modelling for Recommender Systems. En revisión.

## B.5 Estructura de la tesis

La tesis está dividida en seis partes. La primera parte presenta y motiva el problema abordado, así como una revisión del estado del arte en el campo de los Sistemas de Recomendación donde esta tesis se enmarca. La segunda parte describe las diferentes técnicas de evaluación usadas en la literatura de sistemas de recomendación, y provee un análisis de las alternativas de diseño y los sesgos estadísticos que pueden aparecer. La tercera parte hace una revisión de la literatura en predicción de eficacia, propone adaptaciones de este concepto al espacio de los sistemas de recomendación, y evalúa el poder predictivo de estas propuestas. La cuarta parte muestra dos aplicaciones de los predictores de eficacia propuestos. La quinta parte concluye y resume las princi-

pales contribuciones de esta tesis. Información adicional y otros detalles se incluyen en la última parte.

Más detalladamente, los contenidos de esta tesis se distribuyen como sigue:

### **Parte I. Introducción**

- El **Capítulo 1** presenta la motivación, objetivos, contribuciones y publicaciones relacionadas con la tesis.
- El **Capítulo 2** presenta una visión general del estado del arte en sistemas de recomendación, teniendo en cuenta una clasificación de los principales tipos de técnicas. También describe los puntos débiles de las distintas técnicas de recomendación y presenta una amplia gama de métodos híbridos de recomendación que ayudan a superar esas limitaciones.

### **Parte II. Evaluando la Eficacia en los Sistemas de Recomendación**

- El **Capítulo 3** describe las principales métricas y metodologías de evaluación usadas en el campo de los sistemas de recomendación. También describe los conjuntos de datos públicos más habitualmente usados.
- El **Capítulo 4** presenta un análisis y formalización de las diferentes metodologías de evaluación descritas en la literatura. Primero, presenta una caracterización sistemática de las alternativas de diseños experimentales. Después identifica y analiza sesgos estadísticos que aparecen cuando algunas metodologías se aplican a recomendación, y propone dos diseños experimentales alternativos que neutralizan tales sesgos satisfactoriamente.

### **Parte III. Prediciendo la Eficacia en los Sistemas de Recomendación**

- El **Capítulo 5** presenta el problema de predicción de eficacia en Recuperación de Información, incluye un resumen de los principales trabajos en dicha área, tanto en la definición de predictores de eficacia (aplicados a consultas) como en la evaluación de los predictores para estimar su poder predictivo.
- El **Capítulo 6** formula el problema de predicción de eficacia en los Sistemas de Recomendación. Define varios predictores de eficacia basados en tres dimensiones de la recomendación donde analizamos de manera cualitativa el poder predictivo de los predictores.

### **Parte IV. Aplicaciones**

- El **Capítulo 7** propone un marco donde los predictores de eficacia se usan para construir sistemas de recomendación híbridos dinámicos. Evalua estos algoritmos de recomendación en los tres espacios de entrada previamente considerados en la definición de los predictores de eficacia, y usa las distintas alternativas de diseño experimental donde algunos sesgos estadísticos han sido neutralizados.

- El **Capítulo 8** reformula el problema de recomendación basado en usuarios, dando una generalización del mismo como un problema de predicción de eficacia. Investiga cómo adoptar dicha generalización para definir un marco unificado donde podamos realizar un análisis objetivo de la efectividad (poder predictivo) de las funciones de valoración de los vecinos.

## Parte V. Conclusiones

- El **Capítulo 9** concluye con un resumen de las principales contribuciones de esta tesis y una discusión sobre las líneas de investigación futuras.

## Parte VI. Apéndices

- El **Apéndice A** da detalles sobre los métodos propuestos en esta tesis: configuración de los algoritmos de recomendación y los parámetros de los diseños experimentales usados en la evaluación. También reporta estadísticas detalladas sobre los conjuntos de datos utilizados en los experimentos, complementando otros datos incluidos en capítulos previos.
- El **Apéndice B** contiene la traducción a español del Capítulo 1.
- El **Apéndice C** contiene la traducción a español del Capítulo 9.

# Appendix C

## Conclusiones y trabajo futuro

En esta tesis hemos investigado cómo medir y predecir la eficacia de sistemas de recomendación. Hemos analizado y propuesto un conjunto de métodos basados en la adaptación de predictores de eficacia desde el área de Recuperación de Información – principalmente el predictor de claridad de consulta, que captura la ambigüedad de una consulta con respecto a una colección de documentos dada. Hemos definido varios modelos de lenguaje utilizando distintos espacios de probabilidad para capturar los aspectos de los usuarios e ítems implicados en las tareas de recomendación. En este contexto, hemos propuesto y evaluado técnicas novedosas para distintos espacios de entrada extraídas de la Teoría de la Información y la Teoría de Grafos Sociales, usando propiedades sobre las preferencias de usuario así como métricas de grafos, como PageRank sobre la red social del usuario.

Más aún, dado que queremos predecir la eficacia de un sistema de recomendación particular, necesitamos una metodología de evaluación clara con la cual las predicciones de eficacia puedan ser contrastadas. Así, en esta tesis investigamos la metodología de evaluación como parte del problema abordado, donde hemos identificado sesgos estadísticos en la evaluación de la recomendación – a saber, sesgos de dispersión en test y popularidad – los cuales pueden distorsionar las medidas de eficacia, y por tanto, confundir el poder aparente de los métodos de predicción de eficacia. Hemos analizado en profundidad el efecto de dichos sesgos, y hemos propuesto dos diseños experimentales capaces de neutralizar el sesgo de popularidad: una técnica basada en percentiles y un test uniforme. El análisis sistemático de las metodologías de evaluación y las nuevas variantes propuestas permiten una valoración más completa y precisa de la eficiencia de nuestros métodos de predicción de eficacia.

Por otro lado, hemos explotado los métodos propuestos de predicción de eficacia en dos aplicaciones donde se usan para ponderar dinámicamente distintos componentes de un sistema de recomendación, a saber, el ajuste dinámico de recomendaciones híbridas ponderadas, y la ponderación dinámica de las preferencias de los vecinos en filtrado colaborativo basado en usuario. A través de una serie de experimentos empíricos con varios conjuntos de datos y diseños experimentales, hemos encontrado una correspondencia entre el poder predictivo de nuestros predictores de eficacia y la mejora en eficacia de las dos aplicaciones evaluadas.

Presentamos aquí las conclusiones principales obtenidas en este trabajo de investigación. La Sección C.1 muestra un resumen y discusión de nuestras contribuciones, mientras que en la Sección C.2 mostramos vías de investigación que puedan ser abordadas como trabajo futuro.

## C.1 Resumen y discusión de las contribuciones

En las siguientes subsecciones resumimos y discutimos las principales contribuciones de esta tesis, abordando los objetivos enunciados en el Capítulo 1. Estas contribuciones están organizadas de acuerdo a los tres objetivos principales de la tesis. Primero hemos analizado cómo evaluar adecuadamente los sistemas de recomendación para obtener medidas no sesgadas de su eficacia. Segundo hemos propuesto predictores de eficacia que tratan de estimar la eficacia de un método de recomendación. Y tercero hemos usado los predictores de eficacia propuestos para combinar dinámicamente componentes de un sistema de recomendación.

### C.1.1 Análisis de la definición y evaluación de la eficacia en sistemas de recomendación

Hemos analizado distintas alternativas de diseño experimental disponibles en la literatura para sistemas de recomendación, orientados, en particular, a la evaluación basada en rankings, y hemos mostrado que **las hipótesis y condiciones subyacentes al paradigma Cranfield no se pueden asumir en los entornos habituales de recomendación**. Específicamente, hemos detectado sesgos estadísticos que aparecen al aplicar dicho paradigma a la evaluación de sistemas de recomendación. Hemos mostrado que el valor específico de la métrica de evaluación es útil en términos comparativos, pero no tiene un sentido particular en términos absolutos. Hemos mostrado que la precisión decrece linealmente con la dispersión de los ítems relevantes (**sesgo de dispersión**) al usar la metodología de evaluación AR, mientras que no sufre de este sesgo al usar la estrategia 1R.

También hemos observado que un algoritmo de recomendación no personalizado basado en la popularidad de los ítems obtiene valores de eficacia altos, y hemos mostrado y analizado en detalle cómo y por qué esto es debido a un **sesgo de popularidad** en la metodología experimental. Para abordar estos problemas, en esta tesis hemos propuesto **técnicas experimentales novedosas que neutralizan satisfactoriamente el sesgo de popularidad**.

### C.1.2 Definiciones y adaptaciones de predictores de eficacia para sistemas de recomendación

En esta tesis hemos definido y elaborado **predictores de eficacia en el contexto de recomendación**, normalmente tomando al usuario como el objeto de la predicción, pero también considerando los ítems como entradas alternativas para la predicción. Específicamente, hemos adaptado el predictor de eficacia de consulta conocido como *claridad* tomando distintas hipótesis y formulaciones para obtener diferentes variaciones

de los predictores de **claridad del usuario**. También hemos usado conceptos relacionados con la Teoría de la Información como la entropía, métricas de grafos como la centralidad, PageRank y HITS, y otras técnicas heurísticas y específicas del dominio. Hemos definido estos predictores basándonos en tres espacios de entrada para las preferencias de los usuarios: **puntuaciones, registros y redes sociales**. Sobre puntuaciones y registros hemos definido varios modelos de lenguaje y espacios de vocabulario de tal manera que nuestras adaptaciones de claridad capturen distintos aspectos del usuario en una formulación unificada para ambos espacios de entrada. Dentro del mismo marco, hemos introducido la dimensión temporal en los datos de preferencia basados en registros, considerando y elaborando predictores de eficacia basados en tiempo propuestos en trabajos sobre búsqueda ad-hoc previos en el área de RI.

Además, hemos definido **predictores basados en ítems** cuando se usan preferencias basadas en puntuaciones, los cuales intentan estimar la eficacia de los objetos en consideración (siendo más precisos, la eficacia de un sistema de recomendación cuando sugiere dichos ítems). Aquí el principal problema consiste en cómo definir una métrica de eficacia real de manera que un predictor intente estimarla, ya que los ítems no son la entrada principal del proceso de recomendación. Por esta razón, hemos desarrollado metodologías novedosas donde la eficacia de un ítem pueda ser medida, también considerando posibles sesgos que pudieran aparecer cuando usuarios con muchas puntuaciones pueden distorsionar los resultados por razones estadísticas.

Hemos evaluado el acierto predictivo de nuestros métodos calculando la correlación entre la eficacia estimada y la real, siguiendo la práctica estándar en la literatura de predicción de eficacia en RI. De esta manera, hemos usado las metodologías no sesgadas analizadas a lo largo de esta tesis para **comparar cómo se comportan los predictores cuando los sesgos de dispersión y popularidad han sido neutralizados**. Hemos encontrado fuertes valores de correlación confirmando que nuestras técnicas muestran un **poder predictivo significativo**.

### C.1.3 Ponderación dinámica en sistemas de recomendación

La combinación de algoritmos de recomendación es frecuente en la literatura de los Sistemas de Recomendación, en especial lo que se conoce como conjuntos de algoritmos de recomendación (*ensembles*), que son un tipo particular de métodos de recomendación híbrida donde se combinan varios algoritmos, y que actualmente son muy comunes en el área, tal y como se puede comprobar en competiciones recientes (Bennett and Lanning, 2007; Dror et al., 2012). El filtrado colaborativo, una de las técnicas más usadas dentro de la colección de estrategias de recomendación, también se puede ver como una combinación de varias subfunciones de utilidad, cada una correspondiendo a un vecino (en un filtrado basado en usuario). De la misma manera

que los predictores de eficacia en Recuperación de Información se han usado para optimizar la agregación de rankings, nosotros hemos investigado el uso de predictores de eficacia en recomendación para agregar dinámicamente la salida de los algoritmos de recomendación y los vecinos.

Hemos definido un **marco de hibridación dinámica** donde los conjuntos de algoritmos de recomendación pueden beneficiarse de las ponderaciones dinámicas de acuerdo a los predictores de eficacia con los que muestran correlaciones altas. Nuestros resultados indican que correlaciones altas con la eficacia tienden a corresponder con mejoras en los algoritmos de recomendación híbridos dinámicos. Además, los conjuntos dinámicos de recomendación han mostrado mejor eficacia que los conjuntos estáticos para distintas combinaciones de algoritmos y en los tres tipos de predictores de eficacia investigados.

Por otro lado, también hemos propuesto un marco para la **selección y ponderación de vecinos** en sistemas de filtrado colaborativo basados en usuario. Hemos definido predictores y métricas de eficacia de vecinos adaptando e integrando algunos de los métodos de la literatura en recomendación sensibles a la confianza de usuarios. Nuestro marco unifica varias nociones de eficacia de vecinos bajo la misma forma, y presenta un análisis objetivo del poder predictivo de diferentes funciones de valoración de vecinos. Una vez el poder predictivo de estos predictores de vecinos ha sido confirmado, usamos dichos métodos para ponderar la información que proviene de cada vecino de manera dinámica, experimentando con distintas estrategias para la combinación de valores de similitud y pesos de los vecinos. Nuestros experimentos confirman una correspondencia entre los análisis de correlación y los resultados finales de eficacia, en el sentido de que los valores de correlación obtenidos entre los predictores y las métricas de eficacia de vecinos anticipan qué predictores obtendrán mejor eficacia cuando se introduzcan en un algoritmo de filtrado colaborativo basado en usuarios.

## C.2 Trabajo futuro

La predicción de eficacia en recomendación es un área interesante de investigación también desde una perspectiva de negocio, ya que podríamos decidir cuándo entregar las recomendaciones al usuario, evitando disminuir la confianza de los usuarios sobre la relevancia de las sugerencias del sistema. En este sentido, las predicciones pueden dar un control al proveedor de servicios, un control que podría usarse potencialmente de varias maneras, incluyendo una combinación de métodos más general que la que se ha abordado en esta tesis. Independientemente de cualquier aplicación plausible para la industria, y más allá de los logros presentados a lo largo de esta tesis, contemplamos las líneas de investigación futuras que describimos a continuación.

La evaluación de los sistemas de recomendación es aún un objeto de investigación activa en el campo, donde varias cuestiones requieren atención, como el vacío entre experimentos en línea (*online*) y de fuera de línea (*offline*). No obstante, en esta tesis hemos enfocado nuestra investigación en aspectos relacionados con la predicción de eficacia, lo cual requiere un conocimiento más profundo de las metodologías de evaluación utilizadas. De esta manera, podríamos **extender nuestro análisis de las metodologías de evaluación a otras métricas de ranking**, como a aquellas basadas en dos listas de recomendaciones (NDPM y las correlaciones de Spearman y Kendall) o a aquellas adaptadas de Aprendizaje Automático (por ejemplo, el área bajo la curva o AUC en inglés). De esta manera, podríamos encontrar que alguna de estas métricas no está influida por ninguno de los sesgos descritos en el Capítulo 4, o que ninguno de los diseños alternativos propuestos es capaz de neutralizar esos efectos. Como un ejemplo del interés de este tema, recientemente en (Pradel et al., 2012) los autores analizaron los efectos de popularidad sobre la métrica AUC, y encontraron que considerar los datos no puntuados como información negativa durante el entrenamiento podría mejorar la eficacia, pero también podría favorecer a los algoritmos de recomendación basados en popularidad con respecto a los personalizados.

Además, sería beneficioso para nuestra investigación ser capaces de validar la utilidad de las medidas no sesgadas de eficacia con evaluaciones en línea. Esto sería valioso para tener una valoración comparativa con las observaciones fuera de línea que hemos obtenido, así como un conocimiento más profundo de la magnitud por la cual la popularidad puede ser o no una señal ruidosa. Tal estudio de usuario nos ayudaría a determinar los beneficios reales (si los hubiera) de recibir recomendaciones populares, ya que, por ejemplo, por definición estas sugerencias no serían novedosas ni probablemente causales o diversas.

En el Capítulo 6 hemos propuesto varios predictores de eficacia para recomendación basados en los mismos principios de aquellos denominados como predictores pre-búsqueda en Recuperación de Información, como la claridad, donde la salida del algoritmo de búsqueda (o de recomendación en nuestro caso) no es usada por el predictor. Teniendo en cuenta nuestros resultados, las posibilidades para investigar más predictores de eficacia en recomendación son abundantes. En esta línea, varios autores han explotado la **combinación de predictores para obtener valores de correlación mayores y un poder predictivo mayor**, como (Hauff et al., 2009) y (Jones and Diaz, 2007), donde se han usado regresión penalizada y regresión lineal con aprendizaje mediante redes neuronales, respectivamente. En esos trabajos la combinación de predictores de distinta naturaleza mejoró la correlación con respecto a una métrica de evaluación objetivo – en este caso, la precisión promedio. Por ello, creamos que la combinación de predictores puede ser válida también para recomendación, especialmente sabiendo que hemos definido predictores basados en diferentes tipos de datos de los que se espera baja redundancia entre ellos y, por tanto, que dicha

combinación pueda producir correlaciones mayores. Ejemplos de tales combinaciones podrían ser la mezcla de dimensiones sociales y temporales, los predictores temporales basados en ítems, u otras dimensiones contextuales no abordadas en esta tesis.

Más aún, una futura investigación podría **analizar y adaptar también a los sistemas de recomendación predictores de eficacia post-búsqueda** definidos en la literatura de RI, como por ejemplo aquellos basados en el análisis de la distribución de las puntuaciones de los ítems recomendados a cada usuario. Esto podría conseguir predictores con correlaciones más fuertes y, por tanto, con mayor poder predictivo de la eficacia de los algoritmos de recomendación, como ocurre en RI donde los predictores post-búsqueda normalmente obtienen valores de correlación mayores que los pre-búsqueda. La principal limitación de este tipo de predictores es que no pueden ser usados directamente para adaptar la salida de los algoritmos de recomendación, ya que normalmente se requiere la salida completa – es decir, el ranking – para el cálculo de los valores del predictor. Esto obligaría a pensar en distintas aplicaciones donde este tipo de predictores pudieran ser usados en recomendación.

Una dirección particular digna de ser considerada y también relacionada con el Capítulo 6, sería el **uso de técnicas de evaluación alternativas** más allá de las métricas de correlación, como aquellas basadas en el agrupamiento entre los valores de eficacias reales y estimados (ver Sección 5.4.2). En nuestro trabajo nos hemos centrado en el uso de métricas de correlación, principalmente la correlación de Pearson. Estas métricas tienen limitaciones bien conocidas, como su sensibilidad a los valores extremos y correlaciones no significativas cuando se usan un número pequeño de puntos. Por esta razón, se han propuesto otras técnicas para evaluar el poder predictivo de los predictores. Hemos de notar, sin embargo, que el uso de una técnica particular de evaluación debería enfocarse a su aplicación en contextos específicos (Pérez-Iglesias and Araujo, 2010); en particular esto requiere la definición de nuevas aplicaciones para predictores de eficacia que encajen con la métrica de evaluación, lo cual también contemplamos como un potencial trabajo futuro.

También en el Capítulo 6 hemos desarrollado una metodología de evaluación para estimar el valor real de eficacia de los ítems, con el objetivo de evaluar los predictores de ítems propuestos. Esta metodología debería ser validada para **obtener una medida justa de la eficacia del ítem**, lo cual en este momento es aún un problema abierto. De esta manera, seríamos capaces de definir predictores de ítems adicionales para otros espacios de entrada además de las puntuaciones, y de mejorar la capacidad de predicción de los actuales predictores de eficacia de ítems.

En el Capítulo 7 presentamos experimentos sobre combinación dinámica de algoritmos de recomendación en conjunto. Esos experimentos estaban limitados a un único predictor de eficacia por cada par de algoritmos, así que pretendemos extender dichos experimentos con **conjuntos de algoritmos de recomendación donde se consideren dos predictores** para investigar qué condiciones deberían satisfacerse

entre cada par de predictores de manera que se mejore la eficacia del conjunto. Una vía de investigación relacionada a considerar sería el análisis de valores de correlación tales que se obtengan buenos resultados de eficacia en los métodos híbridos dinámicos. Más específicamente, queremos saber si es mejor tener un fuerte valor de correlación en general (en promedio) o un valor medio no tan fuerte pero mejores estimaciones para usuarios particulares, que tendrían un papel significativo en el sistema similar a los usuarios poderosos (*power users*) definidos en (Lathia et al., 2008). En ese punto, se podría realizar un estudio como el presentado en (Hauff et al., 2010), donde simulaciones de predictores con distintos valores de correlación son evaluados, y cuyos efectos sobre la eficacia final en conjuntos de algoritmos de recomendación son comparados.

Además, otra limitación de los experimentos presentados en el Capítulo 7 es que el tamaño de los conjuntos siempre es dos. Pretendemos considerar **conjuntos de algoritmos de recomendación de tamaño N** y a la larga, como se mencionó antes, usar un predictor de eficacia para cada algoritmo de recomendación. Este es un paso natural, pero no trivial hacia la generalización del marco propuesto de conjuntos completos de algoritmos de recomendación. De manera alternativa, podrían usarse técnicas de Aprendizaje Automático para aprender los mejores pesos a usar por cada usuario e ítem en el conjunto. En este caso, se debería investigar un compromiso entre los costes computacionales de cada técnica (aprendizaje automático frente a predictores de eficacia), su poder predictivo y la tendencia a sobreajustar los datos.

Finalmente, en el Capítulo 8 investigamos el problema de ponderación dinámica de vecinos usando predictores de eficacia de vecinos orientados a métricas de error. El trabajo futuro relacionado con este capítulo podría centrarse en la **adaptación de las métricas de eficacia de vecinos usadas en nuestra propuesta hacia métricas de ranking**, tales como la precisión y el *recall*. Como ya hemos discutido, las métricas de error no son la mejor manera de medir la eficacia, aunque se pueden considerar apropiadas en este contexto ya que queremos medir la mejora en la exactitud de nuestras propuestas, y a la vez facilitar comparaciones con el estado del arte en recomendación sensible a la confianza, donde estas métricas son predominantes. Por lo tanto, el uso de métricas de ranking sería una valiosa contribución al campo por sí misma. Además, una vez tuviéramos definidas métricas de eficacia de vecinos basadas en ranking, seríamos capaces de medir la correlación de los predictores de eficacia de vecinos descritos en este capítulo con tales métricas, y analizar en detalle su poder predictivo con métricas de ranking. Idealmente, podríamos obtener un predictor con suficiente poder predictivo usando los dos tipos de métricas de eficacia de vecinos (las basadas en error y en ranking), aunque esto no es fácil de garantizar en general, ya que cada métrica se define para optimizar distintos parámetros y conceptos.



# References

- Adams, R. A. (2007). Music Recommendation using Collaborative Filtering with Similarity Fusion. Master's thesis, Department of Computer Science, The University of York.
- Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749.
- Adomavicius, G., Tuzhilin, A., Berkovsky, S., De Luca, E. W., and Said, A. (2010). Context-awareness in recommender systems: research workshop and movie recommendation challenge. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 385–386, New York, NY, USA. ACM.
- Aggarwal, C. C., Wolf, J. L., Wu, K.-L., and Yu, P. S. (1999). Hatching hatches an egg: a new graph-theoretic approach to collaborative filtering. In *the fifth ACM SIGKDD international conference*, pages 201–212, New York, New York, USA. ACM Press.
- Agrawal, R., Gollapudi, S., Halverson, A., and Ieong, S. (2009). Diversifying search results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, WSDM '09, pages 5–14, New York, NY, USA. ACM.
- Aizawa, A. (2003). An information-theoretic perspective of tf-idf measures. *Information Processing & Management*, 39(1):45–65.
- Albert, R. and Barabási, A. L. (2002). Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97.
- Ali, K. and van Stam, W. (2004). TiVo: making show recommendations using a distributed collaborative filtering architecture. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 394–401, New York, NY, USA. ACM.
- Allan, J. and Raghavan, H. (2002). Using Part-of-speech Patterns to Reduce Query Ambiguity. In *25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 307–314.
- Alvarez, M. M., Yahyaei, S., and Roelleke, T. (2012). Semi-automatic document classification: Exploiting document difficulty. In Baeza Yates, R. A., de Vries, A. P., Zaragoza, H., Cambazoglu, B. B., Murdock, V., Lempel, R., Silvestri, F., Baeza Yates, R. A., de Vries, A. P., Zaragoza, H., Cambazoglu, B. B., Murdock, V., Lempel, R., and Silvestri, F., editors, *ECIR*, volume 7224 of *Lecture Notes in Computer Science*, pages 468–471. Springer.

- Amati, G., Carpineto, C., and Romano, G. (2004). Query Difficulty, Robustness, and Selective Application of Query Expansion. *Advances in Information Retrieval*, pages 127–137.
- Arazy, O., Kumar, N., and Shapira, B. (2009). Improving social recommender systems. *IT Professional*, 11(4):38–44.
- Armstrong, T. G., Moffat, A., Webber, W., and Zobel, J. (2009a). Has adhoc retrieval improved since 1994? In Allan, J., Aslam, J. A., Sanderson, M., Zhai, C., and Zobel, J., editors, *SIGIR*, pages 692–693. ACM.
- Armstrong, T. G., Moffat, A., Webber, W., and Zobel, J. (2009b). Improvements that don't add up: ad-hoc retrieval results since 1998. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 601–610, New York, NY, USA. ACM.
- Aslam, J. A. and Pavlu, V. (2007). Query Hardness Estimation Using Jensen-Shannon Divergence Among Multiple Scoring Functions. In *ECIR*, pages 198–209.
- Aslam, J. A., Pavlu, V., and Yilmaz, E. (2006). A statistical method for system evaluation using incomplete judgments. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 541–548, New York, NY, USA. ACM.
- Baeza-Yates, R. and Ribeiro-Neto, B. (2011). *Modern Information Retrieval: The Concepts and Technology behind Search (2nd Edition)* (ACM Press Books). Addison-Wesley Professional, 2 edition.
- Balabanovic, M. and Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Commun. ACM*, 40(3):66–72.
- Baltrunas, L. and Amatriain, X. (2009). Towards Time-Dependant Recommendation based on Implicit Feedback. In *Context-aware Recommender Systems Workshop at Recsys09*.
- Bao, X., Bergman, L., and Thompson, R. (2009). Stacking recommendation engines with additional meta-features. In *Proceedings of the third ACM conference on Recommender systems*, RecSys '09, pages 109–116, New York, NY, USA. ACM.
- Barbieri, N., Costa, G., Manco, G., and Ortale, R. (2011). Modeling item selection and relevance for accurate recommendations: a bayesian approach. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 21–28, New York, NY, USA. ACM.
- Barman, K. and Dabeer, O. (2010). What is Popular Amongst Your Friends?
- Basu, C., Hirsh, H., and Cohen, W. W. (1998). Recommendation as Classification: Using Social and Content-Based Information in Recommendation. In *AAAI/IJCAI*, pages 714–720.
- Belkin, N. J. and Croft, W. B. (1992). Information filtering and information retrieval: two sides of the same coin? *Commun. ACM*, 35(12):29–38.

- Bell, R. M. and Koren, Y. (2007). Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. In *ICDM '07: Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, pages 43–52, Washington, DC, USA. IEEE Computer Society.
- Bellogín, A. (2009). Performance prediction in recommender systems: application to the dynamic optimisation of aggregative methods. Master's thesis, Escuela Politécnica Superior, Universidad Autónoma de Madrid, Madrid, Spain.
- Bellogín, A., Cantador, I., and Castells, P. (2010). A study of heterogeneity in recommendations for a social music service. In *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, HetRec '10, pages 1–8, New York, NY, USA. ACM.
- Bellogín, A., Cantador, I., Díez, F., Castells, P., and Chavarriaga, E. (2012). An empirical comparison of social, collaborative filtering, and hybrid recommenders. *ACM Transactions on Intelligent Systems and Technology*, to appear.
- Bellogín, A. and Castells, P. (2010). A Performance Prediction Approach to Enhance Collaborative Filtering Performance. In Gurrin, C., He, Y., Kazai, G., Kruschwitz, U., Little, S., Roelleke, T., Rüger, S., and Rijsbergen, editors, *Advances in Information Retrieval*, volume 5993 of *Lecture Notes in Computer Science*, pages 382–393, Berlin, Heidelberg. Springer Berlin / Heidelberg.
- Bellogín, A., Castells, P., and Cantador, I. (2011a). Precision-oriented evaluation of recommender systems: an algorithmic comparison. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 333–336, New York, NY, USA. ACM.
- Bellogín, A., Wang, J., and Castells, P. (2011b). Text Retrieval Methods for Item Ranking in Collaborative Filtering. In Clough, P., Foley, C., Gurrin, C., Jones, G., Kraaij, W., Lee, H., and Mardoch, V., editors, *Advances in Information Retrieval*, volume 6611 of *Lecture Notes in Computer Science*, chapter 30, pages 301–306. Springer Berlin / Heidelberg, Berlin, Heidelberg.
- Ben-Shimon, D., Tsikinovsky, A., Rokach, L., Meisles, A., Shani, G., and Naamani, L. (2007). Recommender System from Personal Social Networks Advances in Intelligent Web Mastering. In Wegrzyn-Wolska, K. and Szczepaniak, P., editors, *Advances in Intelligent Web Mastering*, volume 43 of *Advances in Soft Computing*, chapter 8, pages 47–55. Springer Berlin / Heidelberg, Berlin, Heidelberg.
- Bennett, J. and Lanning, S. (2007). The netflix prize. In *Proceedings of the KDD Cup Workshop 2007*, pages 3–6, New York. ACM.
- Berberich, K., Bedathur, S., Alonso, O., and Weikum, G. (2010). A language modeling approach for temporal information needs. In Gurrin, C., He, Y., Kazai, G., Kruschwitz, U., Little, S., Roelleke, T., Rüger, S., and Rijsbergen, K., editors, *Advances in Information Retrieval*, volume 5993 of *Lecture Notes in Computer Science*, pages 13–25, Berlin, Heidelberg. Springer Berlin / Heidelberg.

- Bernhardsson, E. (2009). Implementing a scalable music recommender system. Master's thesis, School of Engineering Physics, Royal Institute of Technology, Stockholm, Sweden.
- Best, D. J. and Gipps, P. G. (1974). Algorithm AS 71: The upper tail probabilities of kendall's tau. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 23(1):98–100.
- Billsus, D. and Pazzani, M. J. (1998). Learning collaborative information filters. In Shavlik, J. W. and Shavlik, J. W., editors, *ICML*, pages 46–54. Morgan Kaufmann.
- Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022.
- Bollen, D., Knijnenburg, B. P., Willemse, M. C., and Graus, M. (2010). Understanding choice overload in recommender systems. In *RecSys '10: Proceedings of the fourth ACM conference on Recommender systems*, pages 63–70, New York, NY, USA. ACM.
- Bourke, S., McCarthy, K., and Smyth, B. (2011). Power to the people: exploring neighbourhood formations in social recommender system. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 337–340, New York, NY, USA. ACM.
- Bradley, K. and Smyth, B. (2001). Improving Recommendation Diversity. In *Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science*.
- Brants, T., Chen, F., and Tschantaridis, I. (2002). Topic-based document segmentation with probabilistic latent semantic analysis. In *CIKM*, CIKM '02, pages 211–218, New York, NY, USA. ACM.
- Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52.
- Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117.
- Broder, A. (2002). A taxonomy of web search. *SIGIR Forum*, 36(2):3–10.
- Buckley, C. (2004). Topic prediction based on comparative retrieval rankings. In Sanderson, M., Järvelin, K., Allan, J., Bruza, P., Sanderson, M., Järvelin, K., Allan, J., and Bruza, P., editors, *SIGIR*, pages 506–507, New York, NY, USA. ACM.
- Buckley, C., Dimmick, D., Soboroff, I., and Voorhees, E. (2006). Bias and the limits of pooling. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 619–620, New York, NY, USA. ACM.
- Buckley, C., Dimmick, D., Soboroff, I., and Voorhees, E. (2007). Bias and the limits of pooling for large collections. *Information Retrieval*, 10(6):491–508.
- Burke, R. (2002). Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370.

- Burke, R. (2004). Hybrid Recommender Systems with Case-Based Components. pages 91–105.
- Burke, R. (2010). Evaluating the dynamic properties of recommendation algorithms. In *RecSys '10: Proceedings of the fourth ACM conference on Recommender systems*, pages 225–228, New York, NY, USA. ACM.
- Cantador, I. (2008). *Exploiting the conceptual space in hybrid recommender systems: a semantic-based approach*. PhD thesis, Universidad Autonoma de Madrid.
- Cantador, I., Bellogín, A., and Vallet, D. (2010). Content-based recommendation in social tagging systems. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 237–240, New York, NY, USA. ACM.
- Cantador, I., Brusilovsky, P., and Kuflík, T. (2011). Second workshop on information heterogeneity and fusion in recommender systems (HetRec2011). In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 387–388, New York, NY, USA. ACM.
- Cantador, I. and Castells, P. (2006). Multilayered Semantic Social Network Modeling by Ontology-Based User Profiles Clustering: Application to Collaborative Filtering. In *Managing Knowledge in a World of Networks*, pages 334–349.
- Carmel, D. and Yom-Tov, E. (2010). *Estimating the Query Difficulty for Information Retrieval*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers.
- Carmel, D., Yom-Tov, E., Darlow, A., and Pelleg, D. (2006). What makes a query difficult? In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 390–397, New York, NY, USA. ACM.
- Celma, O. (2008). *Music Recommendation and Discovery in the Long Tail*. PhD thesis, Universitat Pompeu Fabra, Barcelona, Spain.
- Celma, O. (2010). *Music Recommendation and Discovery: The Long Tail, Long Tail, and Long Play in the Digital Music Space*. Springer, 1st edition. edition.
- Celma, O. and Cano, P. (2008). From hits to niches?: or how popular artists can bias music recommendation and discovery. In *NETFLIX '08: Proceedings of the 2nd KDD Workshop on Large-Scale Recommender Systems and the Netflix Prize Competition*, pages 1–8, New York, NY, USA. ACM.
- Celma, O. and Herrera, P. (2008). A new approach to evaluating novel recommendations. In *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*, pages 179–186, New York, NY, USA. ACM.
- Chandar, P. and Carterette, B. (2010). Diversification of search results using webgraphs. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 869–870, New York, NY, USA. ACM.

- Clarke, C. L. A., Kolla, M., Cormack, G. V., Vechtomova, O., Ashkan, A., Büttcher, S., and MacKinnon, I. (2008). Novelty and diversity in information retrieval evaluation. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 659–666, New York, NY, USA. ACM.
- Clements, M., de Vries, A., and Reinders, M. J. T. (2009). Exploiting Positive and Negative Graded Relevance Assessments for Content Recommendation. In *Proceedings of the 6th International Workshop on Algorithms and Models for the Web-Graph*, WAW '09, pages 155–166, Berlin, Heidelberg. Springer-Verlag.
- Clements, M., De Vries, A. P., and Reinders, M. J. T. (2010). The task-dependent effect of tags and ratings on social media access. *ACM Trans. Inf. Syst.*, 28.
- Cover, T. M. and Thomas, J. A. (1991). *Elements of Information Theory*. Wiley-Interscience, 99th edition.
- Cremonesi, P., Garzotto, F., Negro, S., Papadopoulos, A., and Turrin, R. (2011). Comparative evaluation of recommender system quality. In *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems*, CHI EA '11, pages 1927–1932, New York, NY, USA. ACM.
- Cremonesi, P., Koren, Y., and Turrin, R. (2010). Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 39–46, New York, NY, USA. ACM.
- Croft, B., Metzler, D., and Strohman, T. (2009). *Search Engines: Information Retrieval in Practice*. Addison Wesley, 1 edition.
- Cronen-Townsend, S., Zhou, Y., and Croft, W. B. (2002). Predicting query performance. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, pages 299–306, New York, NY, USA. ACM.
- Cronen-Townsend, S., Zhou, Y., and Croft, W. B. (2006). Precision prediction based on ranked list coherence. *Information Retrieval*, 9(6):723–755.
- Cummins, R. (2012). Investigating performance predictors using monte carlo simulation and score distribution models. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '12, pages 1097–1098, New York, NY, USA. ACM.
- Cummins, R., Jose, J., and O'Riordan, C. (2011). Improved query performance prediction using standard deviation. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information*, SIGIR '11, pages 1089–1090, New York, NY, USA. ACM.
- Dang, V., Bendersky, M., and Croft, W. B. (2010). Learning to rank query reformulations. In Crestani, F., Maillet, S. M., Chen, H. H., Efthimiadis, E. N., Savoy, J., Crestani, F., Maillet, S. M., Chen, H. H., Efthimiadis, E. N., and Savoy, J., editors, *SIGIR*, SIGIR '10, pages 807–808, New York, NY, USA. ACM.

- Das, A. S., Datar, M., Garg, A., and Rajaram, S. (2007). Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web, WWW '07*, pages 271–280, New York, NY, USA. ACM.
- De Choudhury, M., Mason, W. A., Hofman, J. M., and Watts, D. J. (2010). Inferring relevant social networks from interpersonal communication. In *Proceedings of the 19th international conference on World wide web, WWW '10*, pages 301–310, New York, NY, USA. ACM.
- de Gemmis, M., Lops, P., Semeraro, G., and Basile, P. (2008). Integrating tags in a semantic content-based recommender. In *Proceedings of the 2008 ACM conference on Recommender systems, RecSys '08*, pages 163–170, New York, NY, USA. ACM.
- Demidova, E., Fankhauser, P., Zhou, X., and Nejdl, W. (2010). DivQ: diversification for keyword search over structured databases. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval, SIGIR '10*, pages 331–338, New York, NY, USA. ACM.
- Deshpande, M. and Karypis, G. (2004). Item-based top-N recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177.
- Desrosiers, C. and Karypis, G. (2011). A comprehensive survey of neighborhood-based recommendation methods. In Ricci, F., Rokach, L., Shapira, B., Kantor, P. B., Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, chapter 4, pages 107–144. Springer, Boston, MA.
- Diaz, F. (2007). Performance prediction using spatial autocorrelation. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 583–590, New York, NY, USA. ACM.
- Diaz, F. and Jones, R. (2004). Using temporal profiles of queries for precision prediction. In *SIGIR '04: Proceedings of the 27th annual international conference on Research and development in information retrieval*, pages 18–24. ACM Press.
- Diederich, J. and Iofciu, T. (2006). Finding communities of practice from user profiles based on folksonomies. In Tomadaki, E., Scott, P. J., Tomadaki, E., and Scott, P. J., editors, *EC-TEL Workshops*, volume 213 of *CEUR Workshop Proceedings*. CEUR-WS.org.
- Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271.
- Dror, G., Koenigstein, N., Koren, Y., and Weimer, M. (2012). The yahoo! music dataset and KDD-cup'11. *JMLR Workshop and Conference Proceedings*, 18:3–18.
- Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition.
- Eirinaki, M., Vazirgiannis, M., and Varlamis, I. (2003). SEWeP: using site semantics and a taxonomy to enhance the web personalization process. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108, New York, NY, USA. ACM Press.

- Elahi, M. (2011). Adaptive Active Learning in Recommender Systems. In Konstan, J., Conejo, R., Marzo, J., and Oliver, N., editors, *User Modeling, Adaption and Personalization*, volume 6787 of *Lecture Notes in Computer Science*, chapter 40, pages 414–417. Springer Berlin / Heidelberg, Berlin, Heidelberg.
- Endres, D. M. and Schindelin, J. E. (2003). A new metric for probability distributions. *Information Theory, IEEE Transactions on*, 49(7):1858–1860.
- Fernández, M., Vallet, D., and Castells, P. (2006a). Probabilistic Score Normalization for Rank Aggregation. In *28th European Conference on Information Retrieval (ECIR 2006)*, pages 553–556. Springer Verlag Lecture Notes in Computer Science, Vol. 3936.
- Fernández, M., Vallet, D., and Castells, P. (2006b). Using historical data to enhance rank aggregation. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 643–644, New York, NY, USA. ACM.
- Filippone, M. and Sanguinetti, G. (2010). Information Theoretic Novelty Detection. *Pattern Recognition*, 43(3):805–814.
- Fleder, D. and Hosanagar, K. (2009). Blockbuster Culture’s Next Rise or Fall: The Impact of Recommender Systems on Sales Diversity. *Manage. Sci.*, 55:697–712.
- Foltz, P. W. and Dumais, S. T. (1992). Personalized information delivery: An analysis of information filtering methods. *Commun. ACM*, 35(12):51–60.
- Freeman, L. C. (1977). A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41.
- Gantner, Z., Rendle, S., and Lars, S. T. (2010). Factorization models for context-/time-aware movie recommendations. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa ’10, pages 14–19, New York, NY, USA. ACM.
- Ge, M., Battenfeld, C. D., and Jannach, D. (2010). Beyond accuracy: evaluating recommender systems by coverage and serendipity. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys ’10, pages 257–260, New York, NY, USA. ACM.
- Golbeck, J. (2006). Trust on the world wide web: A survey. *Foundations and Trends in Web Science*, 1(2):131–197.
- Golbeck, J. (2009). Trust and nuanced profile similarity in online social networks. *ACM Trans. Web*, 3(4):1–33.
- Golbeck, J. and Hendler, J. (2006). FilmTrust: movie recommendations using trust in web-based social networks. In *Consumer Communications and Networking Conference, 2006. CCNC 2006. 3rd IEEE*, pages 282–286.
- Goldberg, D., Nichols, D. A., Oki, B. M., and Terry, D. B. (1992). Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70.
- Goldberg, K., Roeder, T., Gupta, D., and Perkins, C. (2001). Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Inf. Retr.*, 4(2):133–151.

- Grivolla, Jourlin, and Mori, D. (2005). Automatic classification of queries by expected retrieval performance. In *Predicting Query Difficulty - Methods and Applications, SIGIR 2005*.
- Gunawardana, A. and Meek, C. (2009). A unified approach to building hybrid recommender systems. In *Proceedings of the third ACM conference on Recommender systems, RecSys '09*, pages 117–124, New York, NY, USA. ACM.
- Gunawardana, A. and Shani, G. (2009). A Survey of Accuracy Evaluation Metrics of Recommendation Tasks. *J. Mach. Learn. Res.*, 10:2935–2962.
- Guo, G., Zhang, J., and Thalmann, D. (2012). A simple but effective method to incorporate trusted neighbors in recommender systems. In Masthoff, J., Mobasher, B., Desmarais, M. C., and Nkambou, R., editors, *User Modeling, Adaptation, and Personalization*, volume 7379 of *Lecture Notes in Computer Science*, pages 114–125, Berlin, Heidelberg. Springer Berlin / Heidelberg.
- Hauff, C. (2010). *Predicting the Effectiveness of Queries and Retrieval Systems*. PhD thesis, Univ. of Twente, Enschede.
- Hauff, C., Azzopardi, L., and Hiemstra, D. (2009). The Combination and Evaluation of Query Performance Prediction Methods. In Boughanem, M., Berrut, C., Mothe, J., Dupuy, C. S., Boughanem, M., Berrut, C., Mothe, J., and Dupuy, C. S., editors, *ECIR*, volume 5478 of *Lecture Notes in Computer Science*, pages 301–312. Springer.
- Hauff, C., Azzopardi, L., Hiemstra, D., and Jong, F. (2010). Query performance prediction: Evaluation contrasted with effectiveness. In Gurrin, C., He, Y., Kazai, G., Kruschwitz, U., Little, S., Roelleke, T., Rüger, S., and Rijsbergen, K., editors, *Advances in Information Retrieval*, volume 5993 of *Lecture Notes in Computer Science*, pages 204–216, Berlin, Heidelberg. Springer Berlin / Heidelberg.
- Hauff, C., Hiemstra, D., and de Jong, F. (2008a). A survey of pre-retrieval query performance predictors. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 1419–1420, New York, NY, USA. ACM.
- Hauff, C., Murdock, V., and Yates, R. B. (2008b). Improved query difficulty prediction for the web. In *Proceeding of the 17th ACM conference on Information and knowledge management, CIKM '08*, pages 439–448, New York, NY, USA. ACM.
- He, B. and Ounis, I. (2004). Inferring Query Performance Using Pre-retrieval Predictors. In *String Processing and Information Retrieval, SPIRE 2004*, pages 43–54.
- He, J., Larson, M., and de Rijke, M. (2008). Using Coherence-Based Measures to Predict Query Difficulty. In Macdonald, C., Ounis, I., Plachouras, V., Ruthven, I., and White, R. W., editors, *Advances in Information Retrieval*, volume 4956 of *Lecture Notes in Computer Science*, chapter 80, pages 689–694. Springer Berlin Heidelberg, Berlin, Heidelberg.

- Herlocker, J., Konstan, J. A., and Riedl, J. (2002). An Empirical Analysis of Design Choices in Neighborhood-Based Collaborative Filtering Algorithms. *Information Retrieval*, 5(4):287–310.
- Herlocker, J. L., Konstan, J. A., Borchers, A., and Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '99, pages 230–237, New York, NY, USA. ACM.
- Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53.
- Hiemstra, D. (1998). A Linguistically Motivated Probabilistic Model of Information Retrieval. In *ECDL '98: Proceedings of the Second European Conference on Research and Advanced Technology for Digital Libraries*, pages 569–584, London, UK. Springer-Verlag.
- Hofmann, T. (2003). Collaborative filtering via gaussian probabilistic latent semantic analysis. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 259–266, New York, NY, USA. ACM.
- Hofmann, T. (2004). Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115.
- Hotho, A., Jäschke, R., Schmitz, C., and Stumme, G. (2006). Information Retrieval in Folksonomies: Search and Ranking. In Sure, Y. and Domingue, J., editors, *The Semantic Web: Research and Applications*, volume 4011 of *Lecture Notes in Computer Science*, chapter 31, pages 411–426. Springer Berlin / Heidelberg, Berlin, Heidelberg.
- Hu, Y., Koren, Y., and Volinsky, C. (2008). Collaborative Filtering for Implicit Feedback Datasets. In *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on*, volume 0, pages 263–272, Washington, DC, USA. IEEE.
- Huang, Z., Zeng, D. D., and Chen, H. (2006). A Unified Recommendation Framework Based on Probabilistic Relational Models. *Social Science Research Network Working Paper Series*.
- Hummel, S., Shtok, A., Raiber, F., Kurland, O., and Carmel, D. (2012). Clarity re-visited. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '12, pages 1039–1040, New York, NY, USA. ACM.
- Hwang, C.-S. and Chen, Y.-P. (2007). Using Trust in Collaborative Filtering Recommendation. In Okuno, H. and Ali, M., editors, *New Trends in Applied Artificial Intelligence*, volume 4570 of *Lecture Notes in Computer Science*, chapter 105, pages 1052–1060. Springer Berlin / Heidelberg, Berlin, Heidelberg.
- Jahrer, M., Töscher, A., and Legenstein, R. (2010). Combining predictions for accurate recommender systems. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 693–702, New York, NY, USA. ACM.

- Jamali, M. and Ester, M. (2009). Using a trust network to improve top-N recommendation. In *RecSys '09: Proceedings of the third ACM conference on Recommender systems*, pages 181–188, New York, NY, USA. ACM.
- Jambor, T. and Wang, J. (2010a). Goal-Driven Collaborative Filtering – A Directional Error Based Approach. In Gurrin, C., He, Y., Kazai, G., Kruschwitz, U., Little, S., Roelleke, T., Rüger, S., and Rijssbergen, K., editors, *Advances in Information Retrieval*, volume 5993, chapter 36, pages 407–419. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Jambor, T. and Wang, J. (2010b). Optimizing multiple objectives in collaborative filtering. In *RecSys '10: Proceedings of the fourth ACM conference on Recommender systems*, pages 55–62, New York, NY, USA. ACM.
- Järvelin, K. and Kekäläinen, J. (2002). Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446.
- Jawaheer, G., Szomszor, M., and Kostkova, P. (2010). Comparison of implicit and explicit feedback from an online music recommendation service. In *Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, HetRec '10, pages 47–51, New York, NY, USA. ACM.
- Jensen, E. C., Beitzel, S. M., Grossman, D., Frieder, O., and Chowdhury, A. (2005). Predicting query difficulty on the web by learning visual clues. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 615–616, New York, NY, USA. ACM.
- Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–20.
- Jones, R. and Diaz, F. (2007). Temporal profiles of queries. *ACM Trans. Inf. Syst.*, 25(3).
- Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632.
- Kohavi, R., Longbotham, R., Sommerfield, D., and Henne, R. M. (2009). Controlled experiments on the web: survey and practical guide. *Data Min. Knowl. Discov.*, 18(1):140–181.
- Kohrs, A. and Merialdo, B. (1999). Clustering for Collaborative Filtering Applications. In *Computational Intelligence for Modelling, Control & Automation (CIMCA '99)*.
- Kompaoré, D., Mothe, J., Baccini, A., and Déjean, S. (2007). Prédiction du sri à utiliser en fonction des critères linguistiques de la requête. In *CORIA*, pages 239–254. Université de Saint-Étienne.
- Konstas, I., Stathopoulos, V., and Jose, J. M. (2009). On social networks and collaborative recommendation. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 195–202, New York, NY, USA. ACM.

- Koren, Y. (2008). Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 426–434, New York, NY, USA. ACM.
- Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37.
- Koren, Y. and Bell, R. M. (2011). Advances in collaborative filtering. In Ricci, F., Rokach, L., Shapira, B., Kantor, P. B., Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, chapter 5, pages 145–186. Springer, Boston, MA.
- Kossinets, G. and Watts, D. J. (2006). Empirical analysis of an evolving social network. *Science*, 311(5757):88–90.
- Kulkarni, A., Teevan, J., Svore, K. M., and Dumais, S. T. (2011). Understanding temporal query dynamics. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 167–176, New York, NY, USA. ACM.
- Kuncheva, L. I. (2004). *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience.
- Kuncheva, L. I. and Whitaker, C. J. (2003). Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy. *Machine Learning*, 51(2):181–207.
- Kwok, K. L., Grunfeld, L., Sun, H. L., and Deng, P. (2004). TREC 2004 Robust Track Experiments Using PIRCS. In *Online Proceedings of 2004 Text REtrieval*.
- Kwon, K., Cho, J., and Park, Y. (2009). Multidimensional credibility model for neighbor selection in collaborative recommendation. *Expert Systems with Applications*, 36(3):7114–7122.
- Kwon, Y. (2008). Improving top-n recommendation techniques using rating variance. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 307–310, New York, NY, USA. ACM.
- Lathia, N. (2010). *Evaluating Collaborative Filtering Over Time*. PhD thesis, University of London, Department of Computer Science, University College London.
- Lathia, N., Hailes, S., and Capra, L. (2008). kNN CF: a temporal social network. In *RecSys '08: Proceedings of the 2008 ACM Conference on Recommender Systems*, pages 227–234, New York, NY, USA. ACM.
- Lathia, N., Hailes, S., Capra, L., and Amatriain, X. (2010). Temporal diversity in recommender systems. In *SIGIR '10: Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 210–217, New York, NY, USA. ACM.
- Lavrenko, V., Allan, J., Deguzman, E., Laflamme, D., Pollard, V., and Thomas, S. (2002). Relevance models for Topic Detection and Tracking. In *Human Language Technology 2002*, pages 104–110.

- Lee, D. H. and Brusilovsky, P. (2009). Reinforcing Recommendation Using Implicit Negative Feedback. In *Proceedings of the 17th International Conference on User Modeling, Adaptation, and Personalization: formerly UM and AH*, volume 5535 of *UMAP '09*, pages 422–427, Berlin, Heidelberg. Springer-Verlag.
- Lee, T., Park, Y., and Park, Y. (2008). A time-based approach to effective recommender systems using implicit feedback. *Expert Systems with Applications*, 34(4):3055–3062.
- Lieberman, H. (1995). Letizia: An agent that assists web browsing. In *IJCAI (1)*, pages 924–929. Morgan Kaufmann.
- Linden, G., Smith, B., and York, J. (2003). Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80.
- Liu, F. and Lee, H. J. (2010). Use of social network information to enhance collaborative filtering performance. *Expert Systems with Applications*, 37(7):4772–4778.
- Liu, K., Fang, B., and Zhang, W. (2010). Speak the same language with your friends: augmenting tag recommenders with social relations. In *Proceedings of the 21st ACM conference on Hypertext and hypermedia*, HT '10, pages 45–50, New York, NY, USA. ACM.
- Lops, P., de Gemmis, M., and Semeraro, G. (2011). Content-based recommender systems: State of the art and trends. In Ricci, F., Rokach, L., Shapira, B., Kantor, P. B., Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, chapter 3, pages 73–105. Springer, Boston, MA.
- Ma, H., King, I., and Lyu, M. R. (2007). Effective missing data prediction for collaborative filtering. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 39–46, New York, NY, USA. ACM.
- Ma, H., King, I., and Lyu, M. R. (2009). Learning to recommend with social trust ensemble. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '09, pages 203–210, New York, NY, USA. ACM.
- Ma, H., Yang, H., Lyu, M. R., and King, I. (2008). SoRec: social recommendation using probabilistic matrix factorization. In *Proceedings of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 931–940, New York, NY, USA. ACM.
- Ma, H., Zhou, D., Liu, C., Lyu, M. R., and King, I. (2011). Recommender systems with social regularization. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 287–296, New York, NY, USA. ACM.
- Macdonald, C., He, B., and Ounis, I. (2005). Predicting Query Performance in Intranet Search. In *Predicting Query Difficulty - Methods and Applications*, SIGIR 2005.
- Magnini, B. and Strapparava, C. (2001). Improving user modelling with Content-Based techniques. In Bauer, M., Gmytrasiewicz, P. J., Vassileva, J., Bauer, M., Gmy-

- trasiewicz, P. J., and Vassileva, J., editors, *User Modeling*, volume 2109 of *Lecture Notes in Computer Science*, pages 74–83. Springer.
- Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press, 1 edition.
- Marlin, B. (2003). Modeling user rating profiles for collaborative filtering. In *In NIPS\*17*.
- Marlin, B. M., Zemel, R. S., Roweis, S., and Slaney, M. (2007). Collaborative filtering and the missing at random assumption. In *Proc. of the 23rd Conference on Uncertainty in Artificial Intelligence*.
- Martinez, A., Arias, J., Vilas, A., Garcia, and Lopez (2009). What's on TV tonight? An efficient and effective personalized recommender system of TV programs. *Consumer Electronics, IEEE Transactions on*, 55(1):286–294.
- Marx, P., Thurau, T. H., and Marchand, A. (2010). Increasing consumers' understanding of recommender results: a preference-based hybrid algorithm with strong explanatory power. In *RecSys '10: Proceedings of the fourth ACM conference on Recommender systems*, pages 297–300, New York, NY, USA. ACM.
- Massa, P. and Avesani, P. (2004). Trust-Aware Collaborative Filtering for Recommender Systems. In Meersman, R. and Tari, Z., editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, volume 3290 of *Lecture Notes in Computer Science*, chapter 31, pages 492–508. Springer Berlin / Heidelberg, Berlin, Heidelberg.
- Massa, P. and Avesani, P. (2007a). Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, RecSys '07, pages 17–24, New York, NY, USA. ACM.
- Massa, P. and Avesani, P. (2007b). Trust metrics on controversial users: balancing between tyranny of the majority and echo chambers. *International Journal on Semantic Web and Information Systems*.
- Massa, P. and Bhattacharjee, B. (2004). Using trust in recommender systems: An experimental analysis. In Jensen, C. D., Poslad, S., Dimitrakos, T., Jensen, C. D., Poslad, S., and Dimitrakos, T., editors, *iTrust*, volume 2995 of *Lecture Notes in Computer Science*, pages 221–235, Berlin, Heidelberg. Springer.
- McLaughlin, M. R. and Herlocker, J. L. (2004). A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, pages 329–336, New York, NY, USA. ACM.
- McNee, S. M., Riedl, J., and Konstan, J. A. (2006). Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06 extended abstracts on Human factors in computing systems*, CHI EA '06, pages 1097–1101, New York, NY, USA. ACM.

- Melucci, M. (2009). Weighted rank correlation in information retrieval evaluation. In Lee, G. G., Song, D., Lin, C. Y., Aizawa, A. N., Kuriyama, K., Yoshioka, M., Sakai, T., Lee, G. G., Song, D., Lin, C. Y., Aizawa, A. N., Kuriyama, K., Yoshioka, M., and Sakai, T., editors, *AIRS*, volume 5839 of *Lecture Notes in Computer Science*, pages 75–86. Springer.
- Michlmayr, E. and Cazer, S. (2007). Learning user profiles from tagging data and leveraging them for personal(ized) information access. In *Tagging and Metadata for Social Information Organization Workshop in conjunction with the 16th International World Wide Web Conference*.
- Milgram, S. (1967). The small world problem. *Psychology Today*, 1:61–67.
- Mooney, R. J. and Roy, L. (2000). Content-based book recommending using learning for text categorization. In *Proceedings of the fifth ACM conference on Digital libraries*, DL '00, pages 195–204, New York, NY, USA. ACM.
- Mothe, J. and Tanguy, L. (2005). Linguistic features to predict query difficulty. In *Predicting Query Difficulty - Methods and Applications*, SIGIR 2005.
- Newman, M. E. J. (2003). Ego-centered networks and the ripple effect. *Social Networks*, 25(1):83–95.
- Oard, D. and Kim, J. (1998). Implicit Feedback for Recommender Systems. In *in Proceedings of the AAAI Workshop on Recommender Systems*, pages 81–83.
- O'Connor, M. and Herlocker, J. (1999). Clustering Items for Collaborative Filtering. In *ACM SIGIR Workshop on Recommender Systems*.
- O'Donovan, J. and Smyth, B. (2005). Trust in recommender systems. In *Proceedings of the 10th international conference on Intelligent user interfaces*, IUI '05, pages 167–174, New York, NY, USA. ACM.
- O'Madadhain, J., Fisher, D., White, S., and Boey, Y. B. (2003). The JUNG (java universal Network/Graph) framework. Technical Report UCI-ICS 03-17, University of California.
- Onuma, K., Tong, H., and Faloutsos, C. (2009). TANGENT: a novel, 'Surprise me', recommendation algorithm. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 657–666, New York, NY, USA. ACM.
- Parra, D. and Amatriain, X. (2011). Walk the Talk. In Konstan, J. A., Conejo, R., Marzo, J. L., and Oliver, N., editors, *User Modeling, Adaption and Personalization*, volume 6787 of *Lecture Notes in Computer Science*, pages 255–268, Berlin, Heidelberg. Springer Berlin / Heidelberg.
- Pavlov, D., Manavoglu, E., Pennock, D. M., and Giles, C. L. (2004). Collaborative Filtering with Maximum Entropy. *IEEE Intelligent Systems*, 19(6):40–48.
- Pazzani, M. and Billsus, D. (1997). Learning and Revising User Profiles: The Identification of Interesting Web Sites. *Machine Learning*, 27(3):313–331.

- Pazzani, M. and Billsus, D. (2007). Content-Based Recommendation Systems The Adaptive Web. In Brusilovsky, P., Kobsa, A., and Nejdl, W., editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, chapter 10, pages 325–341. Springer Berlin / Heidelberg, Berlin, Heidelberg.
- Pazzani, M. J. (1999). A Framework for Collaborative, Content-Based and Demographic Filtering. *Artificial Intelligence Review*, 13(5):393–408.
- Pazzani, M. J., Muramatsu, J., and Billsus, D. (1996). Syskill & webert: Identifying interesting web sites. In Clancey, W. J., Weld, D. S., Clancey, W. J., and Weld, D. S., editors, *AAAI/LAAI, Vol. 1*, pages 54–61. AAAI Press / The MIT Press.
- Pérez Iglesias, J. (2012). *Predicción del rendimiento de consultas basado en rankings de documentos y nuevo marco de evaluación*. PhD thesis, Universidad Nacional de Educación a Distancia.
- Pérez-Iglesias, J. and Araujo, L. (2009). Ranking List Dispersion as a Query Performance Predictor. pages 371–374.
- Pérez-Iglesias, J. and Araujo, L. (2010). Evaluation of Query Performance Prediction Methods by Range. In Chavez, E. and Lonardi, S., editors, *String Processing and Information Retrieval*, volume 6393 of *Lecture Notes in Computer Science*, chapter 23, pages 225–236–236. Springer Berlin / Heidelberg, Berlin, Heidelberg.
- Plachouras, V., He, B., and Ounis, I. (2004). University of Glasgow at TREC 2004: Experiments in Web, Robust, and Terabyte Tracks with Terrier. In Voorhees, E. M., Buckland, L. P., Voorhees, E. M., and Buckland, L. P., editors, *TREC*, volume Special Publication 500-261. National Institute of Standards and Technology (NIST).
- Plachouras, V., Ounis, I., van Rijsbergen, C. J., and Cacheda, F. (2003). University of Glasgow at the Web Track: Dynamic Application of Hyperlink Analysis using the Query Scope. In *TREC*, pages 646–652.
- Ponte, J. M. and Croft, W. B. (1998). A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR '98*, pages 275–281, New York, NY, USA. ACM.
- Pradel, B., Usunier, N., and Gallinari, P. (2012). Ranking with non-random missing ratings: influence of popularity and positivity on evaluation metrics. In *Proceedings of the sixth ACM conference on Recommender systems, RecSys '12*, pages 147–154, New York, NY, USA. ACM.
- Pu, P., Chen, L., and Hu, R. (2012). Evaluating recommender systems from the user's perspective: survey of the state of the art. *User Modeling and User-Adapted Interaction*, 22(4):317–355.
- Radlinski, F., Bennett, P. N., Carterette, B., and Joachims, T. (2009). Redundancy, diversity and interdependent document relevance. *SIGIR Forum*, 43(2):46–52.

- Radlinski, F., Kleinberg, R., and Joachims, T. (2008). Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 784–791, New York, NY, USA. ACM.
- Rafiei, D., Bharat, K., and Shukla, A. (2010). Diversifying web search results. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 781–790, New York, NY, USA. ACM.
- Rafter, R., O'Mahony, M., Hurley, N., and Smyth, B. (2009). What Have the Neighbours Ever Done for Us? A Collaborative Filtering Perspective. In Houben, G.-J., McCalla, G., Pianesi, F., and Zancanaro, M., editors, *User Modeling, Adaptation, and Personalization*, volume 5535 of *Lecture Notes in Computer Science*, chapter 36, pages 355–360. Springer Berlin / Heidelberg, Berlin, Heidelberg.
- Renda, E. M. and Straccia, U. (2003). Web metasearch: rank vs. score based rank aggregation methods. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 841–846, New York, NY, USA. ACM Press.
- Rennie, J. D. M. and Srebro, N. (2005). Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 713–719, New York, NY, USA. ACM.
- Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina. ACM.
- Ricci, F., Rokach, L., and Shapira, B. (2011). Introduction to recommender systems handbook. In Ricci, F., Rokach, L., Shapira, B., Kantor, P. B., Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, chapter 1, pages 1–35. Springer, Boston, MA.
- Rich, E. (1979). User modeling via stereotypes. *Cognitive Science*, 3:335–366.
- Roelleke, T. and Wang, J. (2008). TF-IDF uncovered: a study of theories and probabilities. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 435–442, New York, NY, USA. ACM.
- Rojasattarat, E. and Soonthornphisaj, N. (2003). Hybrid Recommendation: Combining Content-Based Prediction and Collaborative Filtering. In *Intelligent Data Engineering and Automated Learning*, pages 337–344.
- Said, A., Berkovsky, S., and De Luca, E. W. (2010). Putting things in context: Challenge on Context-Aware movie recommendation. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa '10, pages 2–6, New York, NY, USA. ACM.
- Said, A., Berkovsky, S., De Luca, E. W., and Hermanns, J. (2011). Challenge on context-aware movie recommendation: CAMRa2011. In *Proceedings of the fifth ACM*

- conference on Recommender systems*, RecSys '11, pages 385–386, New York, NY, USA. ACM.
- Salakhutdinov, R., Mnih, A., and Hinton, G. E. (2007). Restricted boltzmann machines for collaborative filtering. In Ghahramani, Z. and Ghahramani, Z., editors, *ICML*, volume 227 of *ACM International Conference Proceeding Series*, pages 791–798, New York, NY, USA. ACM.
- Salter, J. and Antonopoulos, N. (2006). CinemaScreen Recommender Agent: Combining Collaborative and Content-Based Filtering. *IEEE Intelligent Systems*, 21(1):35–41.
- Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA. ACM.
- Schein, A., Popescul, A., Ungar, L., and Pennock, D. (2002). Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2002)*, pages 253–260.
- Schein, A. I., Popescul, A., Ungar, L. H., and Pennock, D. M. (2001). Generative models for cold-start recommendations. In *Proceedings of the 2001 SIGIR Workshop on Recommender Systems*.
- Semeraro, G., Degennaris, M., Lops, P., and Basile, P. (2007). Combining learning and word sense disambiguation for intelligent user profiling. In Veloso, M. M. and Veloso, M. M., editors, *IJCAI*, pages 2856–2861.
- Shani, G. and Gunawardana, A. (2011). Evaluating Recommendation Systems. In Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B., editors, *Recommender Systems Handbook*, chapter 8, pages 257–297. Springer US, Boston, MA.
- Shardanand, U. and Maes, P. (1995). Social information filtering: Algorithms for automating "word of mouth". In Katz, I. R., Mack, R. L., Marks, L., Rosson, M. B., Nielsen, J., Katz, I. R., Mack, R. L., Marks, L., Rosson, M. B., and Nielsen, J., editors, *CHI, CHI '95*, pages 210–217, New York, NY, USA. ACM/Addison-Wesley.
- Shepitsen, A., Gemmell, J., Mobasher, B., and Burke, R. (2008). Personalized Recommendation in Social Tagging Systems using Hierarchical Clustering. In *Proceedings of the 2008 ACM Conference on Recommender Systems (RecSys 2008)*, RecSys '08, pages 259–266, New York, NY, USA. ACM.
- Shtok, A., Kurland, O., and Carmel, D. (2009). Predicting query performance by Query-Drift estimation. In *Advances in Information Retrieval Theory*, pages 305–312.
- Shtok, A., Kurland, O., and Carmel, D. (2010). Using statistical decision theory and relevance models for query-performance prediction. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 259–266, New York, NY, USA. ACM.

- Snedecor, G. W. and Cochran, W. G. (1989). *Statistical Methods*. Iowa State University Press, 8 edition.
- Soboroff, I. (2004). On evaluating web search with very few relevant documents. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, pages 530–531, New York, NY, USA. ACM.
- Steck, H. (2011). Item popularity and recommendation accuracy. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 125–132, New York, NY, USA. ACM.
- Su, L. T. (1992). Evaluation measures for interactive information retrieval. *Information Processing & Management*, 28(4):503–516.
- Sun, A. and Bhowmick, S. S. (2009). Image tag clarity: in search of visual-representative tags for social images. In *Proceedings of the first SIGMM workshop on Social media*, WSM '09, pages 19–26, New York, NY, USA. ACM.
- Sun, A. and Datta, A. (2009). On stability, clarity, and co-occurrence of Self-Tagging. In Baeza Yates, R. A., Boldi, P., Ribeiro Neto, B. A., Cambazoglu, B. B., Baeza Yates, R. A., Boldi, P., Ribeiro Neto, B. A., and Cambazoglu, B. B., editors, *WSDM (Late Breaking-Results)*. ACM.
- Teevan, J., Dumais, S. T., and Liebling, D. J. (2008). To personalize or not to personalize: modeling queries with variation in user intent. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 163–170, New York, NY, USA. ACM.
- Tomlinson, S. (2005). European ad hoc retrieval experiments with hummingbird SearchServer TM at. In *CLEF 2005. Working Notes for the CLEF 2005 Workshop*.
- Townsend, S. C., Zhou, Y., and Croft, B. W. (2004). A framework for selective query expansion. In Grossman, D., Gravano, L., Zhai, C., Herzog, O., Evans, D. A., Grossman, D., Gravano, L., Zhai, C., Herzog, O., and Evans, D. A., editors, *CIKM*, pages 236–237. ACM.
- van Rijsbergen, C. J. (1989). Towards an information logic. *SIGIR Forum*, 23(SI):77–86.
- van Setten, M. (2005). *Supporting people in finding information: hybrid recommender systems and goal-based structuring*. PhD thesis, University of Twente, Enschede.
- Vargas, S. and Castells, P. (2011). Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys '11, pages 109–116, New York, NY, USA. ACM.
- Vinay, V., Cox, I. J., Milic-Frayling, N., and Wood, K. (2006). On ranking the effectiveness of searches. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 398–404, New York, NY, USA. ACM Press.
- Voorhees, E. M. (2002a). Overview of trec 2002. In *TREC*.

- Voorhees, E. M. (2002b). The philosophy of information retrieval evaluation evaluation of Cross-Language information retrieval systems. In Peters, C., Braschler, M., Gonzalo, J., and Kluck, M., editors, *Evaluation of Cross-Language Information Retrieval Systems*, volume 2406 of *Lecture Notes in Computer Science*, chapter 34, pages 143–170. Springer Berlin / Heidelberg, Berlin, Heidelberg.
- Voorhees, E. M. (2005a). Overview of the TREC 2004 robust retrieval track. In *Proceedings of the Thirteenth Text REtrieval Conference, TREC 2004*, pages 70–79.
- Voorhees, E. M. (2005b). The TREC robust retrieval track. *SIGIR Forum*, 39(1):11–20.
- Voorhees, E. M. (2006). The TREC 2005 robust track. *SIGIR Forum*, 40(1):41–48.
- Voorhees, E. M. and Harman, D. K., editors (2005). *TREC: Experiment And Evaluation in Information Retrieval*. MIT Press, Cambridge, MA.
- Walter, F. E., Battiston, S., and Schweitzer, F. (2009). Personalised and dynamic trust in social networks. In *Proceedings of the third ACM conference on Recommender systems, RecSys '09*, pages 197–204, New York, NY, USA. ACM.
- Wang, J. (2009). Language Models of Collaborative Filtering. In Lee, G. G., Song, D., Lin, C.-Y., Aizawa, A., Kuriyama, K., Yoshioka, M., and Sakai, T., editors, *Information Retrieval Technology*, volume 5839, chapter 19, pages 218–229. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Wang, J., de Vries, A., and Reinders, M. (2006a). A User-Item Relevance Model for Log-Based Collaborative Filtering. In Lalmas, M., MacFarlane, A., Rüger, S., Tombros, A., Tsikrika, T., and Yavlinsky, A., editors, *Advances in Information Retrieval*, volume 3936 of *Lecture Notes in Computer Science*, chapter 5, pages 37–48–48. Springer Berlin / Heidelberg, Berlin, Heidelberg.
- Wang, J., de Vries, A. P., and Reinders, M. J. T. (2006b). Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR '06*, pages 501–508, New York, NY, USA. ACM.
- Wang, J., de Vries, A. P., and Reinders, M. J. T. (2008a). Unified relevance models for rating prediction in collaborative filtering. *ACM Trans. Inf. Syst.*, 26(3):1–42.
- Wang, J., Robertson, S., de Vries, A., and Reinders, M. (2008b). Probabilistic relevance ranking for collaborative filtering. *Information Retrieval*, 11(6):477–497.
- Wang, X., Fang, H., and Zhai, C. (2008c). A study of methods for negative relevance feedback. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR '08*, pages 219–226, New York, NY, USA. ACM.
- Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442.

- Weng, J., Miao, C., and Goh, A. (2006). Improving collaborative filtering with trust-based metrics. In Haddad, H. and Haddad, H., editors, *SAC*, pages 1860–1864, New York, NY, USA. ACM.
- Weng, L. T., Xu, Y., Li, Y., and Nayak, R. (2007). Improving Recommendation Novelty Based on Topic Taxonomy. *Web Intelligence and Intelligent Agent Technology, International Conference on*, 0:115–118.
- Xiang, L., Yuan, Q., Zhao, S., Chen, L., Zhang, X., Yang, Q., and Sun, J. (2010). Temporal recommendation on graphs via long- and short-term preference fusion. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD ’10, pages 723–732, New York, NY, USA. ACM.
- Xin, Y. and Steck, H. (2011). Multi-value probabilistic matrix factorization for IP-TV recommendations. In *Proceedings of the fifth ACM conference on Recommender systems*, RecSys ’11, pages 221–228, New York, NY, USA. ACM.
- Xue, G. R., Lin, C., Yang, Q., Xi, W., Zeng, H. J., Yu, Y., and Chen, Z. (2005). Scalable collaborative filtering using cluster-based smoothing. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR ’05, pages 114–121, New York, NY, USA. ACM.
- Yao, Y. Y. (1995). Measuring retrieval effectiveness based on user preference of documents. *JASIS*, 46(2):133–145.
- Yilmaz, E., Aslam, J. A., and Robertson, S. (2008). A new rank correlation coefficient for information retrieval. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR ’08, pages 587–594, New York, NY, USA. ACM.
- Yom-Tov, E., Fine, S., Carmel, D., and Darlow, A. (2005a). Learning to estimate query difficulty: including applications to missing content detection and distributed information retrieval. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR ’05, pages 512–519, New York, NY, USA. ACM.
- Yom-Tov, E., Fine, S., Carmel, D., and Darlow, A. (2005b). Metasearch and Federation using Query Difficulty Prediction. In *Predicting Query Difficulty - Methods and Applications, SIGIR 2005*.
- Yu, K., Schwaighofer, A., Tresp, V., Ma, W. Y., and Zhang, H. (2003). Collaborative Ensemble Learning: Combining Collaborative and Content-Based Information Filtering via Hierarchical Bayes. In Meek, C., Kj, U., Meek, C., and Kj, U., editors, *UAI*, pages 616–623. Morgan Kaufmann.
- Zar, J. H. (1972). Significance testing of the spearman rank correlation coefficient. *Journal of the American Statistical Association*, 67(339):578–580.
- Zhang, M. and Hurley, N. (2008). Avoiding monotony: improving the diversity of recommendation lists. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys ’08, pages 123–130, New York, NY, USA. ACM.

- Zhang, M. and Hurley, N. (2009). Statistical Modeling of Diversity in Top-N Recommender Systems. *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, 1:490–497.
- Zhang, Y., Callan, J., and Minka, T. (2002). Novelty and redundancy detection in adaptive filtering. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, pages 81–88, New York, NY, USA. ACM.
- Zhao, Y., Scholer, F., and Tsegay, Y. (2008). Effective Pre-retrieval Query Performance Prediction Using Similarity and Variability Evidence. In Macdonald, C., Ounis, I., Plachouras, V., Ruthven, I., and White, R. W., editors, *Advances in Information Retrieval*, volume 4956 of *Lecture Notes in Computer Science*, chapter 8, pages 52–64. Springer Berlin Heidelberg, Berlin, Heidelberg.
- Zhou, T., Kuscsik, Z., Liu, J.-G., Medo, M., Wakeling, J. R., and Zhang, Y.-C. (2010). Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515.
- Zhou, Y. (2007). *Retrieval Performance Prediction and Document Quality*. PhD thesis, University of Massachusetts.
- Zhou, Y. and Croft, W. B. (2006). Ranking robustness: a novel framework to predict query performance. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, CIKM '06, pages 567–574, New York, NY, USA. ACM.
- Zhou, Y. and Croft, W. B. (2007). Query performance prediction in web search environments. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, pages 543–550, New York, NY, USA. ACM.
- Ziegler, C. N. and Lausen, G. (2004). Analyzing correlation between trust and user similarity in online communities. In Jensen, C. D., Poslad, S., Dimitrakos, T., Jensen, C. D., Poslad, S., and Dimitrakos, T., editors, *iTrust*, volume 2995 of *Lecture Notes in Computer Science*, pages 251–265. Springer.
- Zimdars, A., Chickering, D. M., and Meek, C. (2001). Using Temporal Data for Making Recommendations. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, UAI '01, pages 580–588, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Zitnick, C. L. and Kanade, T. (2004). Maximum entropy for collaborative filtering. In Chickering, D. M., Halpern, J. Y., Chickering, D. M., and Halpern, J. Y., editors, *UAI*. AUAI Press.