Recommendation Systems

Content-Based Recommendations
Collaborative Filtering
Hybrid Systems

Professor Wei-Min Shen

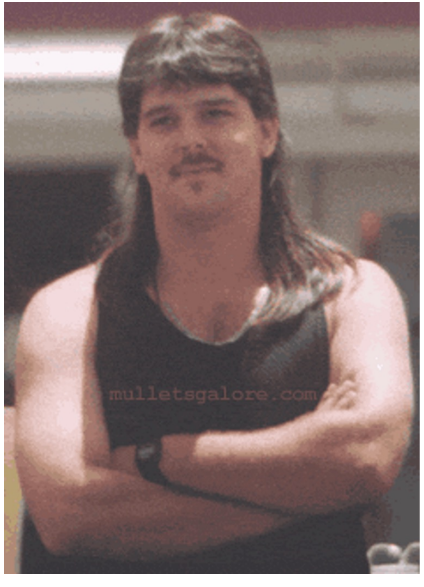
University of Southern California

Thanks for source slides and material to:
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets
http://www.mmds.org

Additional Reading

- Overview articles: Wikipedia pages on Recommender Systems and Collaborative Filtering:
  - http://en.wikipedia.org/wiki/Recommender_system
  - http://en.wikipedia.org/wiki/Collaborative_filtering
- Motivation: Article on The Long Tail
  - http://www.wired.com/wired/archive/12.10/tail.html
- Recommender Systems, Prem Melville and Vikas Sindhwani, Encyclopedia of Machine Learning, 2010
  - http://www.prem-melville.com/publications/recommender-systems-eml2010.pdf
- A Survey of Collaborative Filtering Techniques
  - http://dl.acm.org/citation.cfm?id=1722966

Example: Recommender Systems



- **Customer X**
  - Buys Metallica CD
  - Buys Megadeth CD

- **Customer Y**
  - Does search on Metallica
  - Recommender system suggests Megadeth from data collected about customer **X**

Recommendations

Search

Recommendations

Items

Products, web sites, blogs, news items, …

Examples:

amazon.com

PANDORA

StumbleUpon

del.icio.us

NETFLIX

movielens
helping you find the *right* movies

last·fm
the social music revolution

Google News

YouTube

XBOX LIVE

一、1. Motivation: The Long Tail

From Scarcity to Abundance

- **Shelf space is a scarce commodity for traditional retailers**
  - Also: TV networks, movie theaters,…
- **Web enables near-zero-cost dissemination of information about products**
  - From scarcity to abundance
- **More choice necessitates better filters**
  - Recommendation engines
  - How **Into Thin Air** made **Touching the Void** a bestseller: http://www.wired.com/wired/archive/12.10/tail.html

Touching the Void

- **In 1988,** Joe Simpson wrote a book called *Touching the Void*, a harrowing account of near death in the Peruvian Andes.
  - It got only a modest success, it was soon forgotten.
- A decade later, Jon Krakauer wrote *Into Thin Air*, another book about a mountain-climbing tragedy
  - which became a publishing sensation.
- Suddenly *Touching the Void* started to sell again.
- Now *Touching the Void* outsells *Into Thin Air* more than two to one.

The Long Tail

Sources: Erik Brynjolfsson and Jeffrey Hu, MIT, and Michael Smith, Carnegie Mellon; Barnes & Noble; Netflix; RealNetworks

Source: Chris Anderson (2004)

Change in thinking compared with online stores

- "What percentage of the top 10,000 titles in any online media store (Netflix, iTunes, Amazon, or any other) will rent or sell at least once a month?"
- Most people guess 20 percent
  - 80-20 rule, also known as Pareto's principle (1896)
  - Only 20 percent of major studio films, TV shows, books, etc. will be hits
- The right answer: 99 percent
  - Demand for nearly every one of those top 10,000 titles

Counterintuitive to old way of thinking

- The 20 percent rule in the entertainment industry is about ***hits***, not sales of any sort
  - **Hit-driven mindset**: think that if something isn't a hit, it won't make money
  - Makes sense with scarce shelf space in a retail store
  - iTunes, Amazon, and Netflix: discovered that **"misses" usually make money, too**
  - And because there are so many more of them, that money can add up quickly to a huge new market
- Industry has a poor sense of what people want
  - Turns out that people like a wide range of things when they are easily available

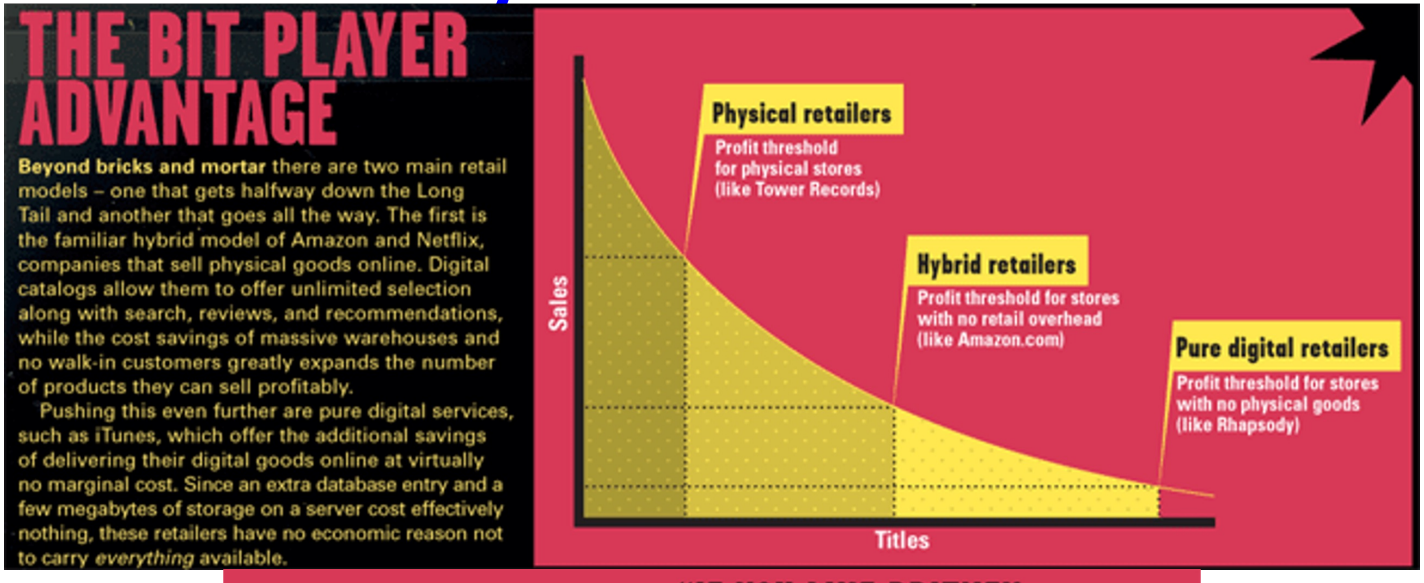
2. Rules of thumb

- **Rule 1: Make everything available**
  - Embrace underserved markets, niches (e.g., obscure video genres)
  - What matters is not where customers are, or how many of them are seeking a particular title, but that some number of them exist, anywhere
  - As a result, almost anything is worth offering on the chance it will find a buyer
  - Example:
    - "Put it on the craiglist!"
    - "One man's junk is another's treasure"!

Rules of thumb (cont.)

- **Rule 2: Lower costs**
  - Have taken away the unnecessary costs of the retail channel: manufacturing, distribution, and retail overheads
  - Leaves the costs of finding, making, and marketing content
  - Ensure that the people on the creative and business side still make money
- **Rule 3: Help users find new content, easily**
  - Edited recommendations
  - Content-based recommendations
  - Collaborative filtering: uses browsing and purchasing patterns of users to guide those who follow them
  - "Customers who bought this also bought ..."
  - **Use recommendations to drive demand down the Long Tail**

Physical vs. Online



http://www.wired.com/wired/archive/12.10/tail.html

3. Types of Recommendations

- **Editorial and hand curated** old way
  - List of favorites
  - Lists of "essential" items
- **Simple aggregates**
  - Top 10, Most Popular, Recent Uploads
- **Tailored to individual users** now
  - Amazon, Netflix, …

Formal Model

- ***X*** = set of **Customers**
- ***S*** = set of **Items**
- Users have preferences for certain items
- **Want to extract preferences from data**

- **Utility function** ***u***: ***X*** × ***S*** => ***R***
  - ***R*** = set of ratings
  - ***R*** is a totally ordered set
    - e.g., **0-5** stars, real number in **[0,1]**

2. Utility Matrix

- For each user-item pair, value represents degree of preference of that user for that item (e.g., rating)
- Matrix is sparse (most entries unknown)

S

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.2	
Bob		0.5		0.3
Carol	0.2		1	
David				0.4

X

3. Predictions

- Goal of a Recommendation System is to **predict the blanks in the utility matrix**
  - Would Alice like Pirates?
  - Would David like Avatar?
- Not necessary to predict every blank entry
- **Want to discover some entries in each row that are likely to be high**
- **Usually recommend a few items the user should value highly**
  - **Most likely to generate additional revenue**

Key Problems

- **(1) Gathering "known" ratings for matrix**
  - How to collect the data in the utility matrix
- **(2) Extrapolate unknown ratings from the known ones**
  - Mainly interested in **high unknown ratings**
    - We are not interested in knowing what you don't like but what you like
    - To generate revenue
- **(3) Evaluating extrapolation methods**
  - How to measure success/performance of recommendation methods

(1) Gathering Ratings

- **Explicit**
  - Ask people to rate items
  - Doesn't work well in practice – people can't be bothered
- **Implicit**
  - **Learn ratings from user actions**
    - E.g., purchase implies high rating
  - What about low ratings?

prediction

(2) Extrapolating Utilities

- **Key problem:** Utility matrix ***U*** is **sparse**
  - Most people have not rated most items
  - **Cold start:**
    - **New items have no ratings**
    - **New users have no history**

- **How to extrapolate missing entries?**

Three Approaches to Recommendation Systems

- **1) Content-based**
  - Use characteristics of an item
  - Recommend items that have similar content to items user liked in the past
  - Or items that match predefined attributes of the user
- **2) Collaborative filtering**
  - Build a model from a user's past behavior (items previously purchased or rated) and similar decisions made by other users
  - Use the model to predict items that the user may like
  - Collaborative: suggestions made to a user utilize information across the entire user base
- **3) Hybrid approaches**

Content-based Recommendations (Item-based or user-based)

- **Main idea: Recommend items to customer *x* that are similar to previous items rated highly by *x***
  - ***Requires characterizing the content of items in some way***

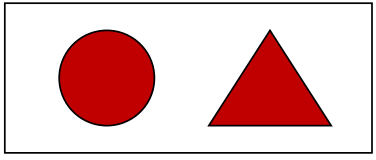
Examples:

- **Movie recommendations**
  - Recommend movies with same actor(s), director, genre, …
- **Websites, blogs, news**
  - Recommend other sites with "similar" content

Plan of Action

likes

Item profiles

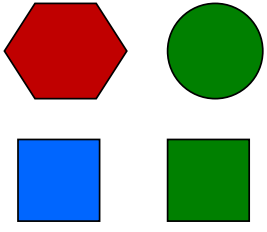


build

Red
Circles
Triangles

User profile

match

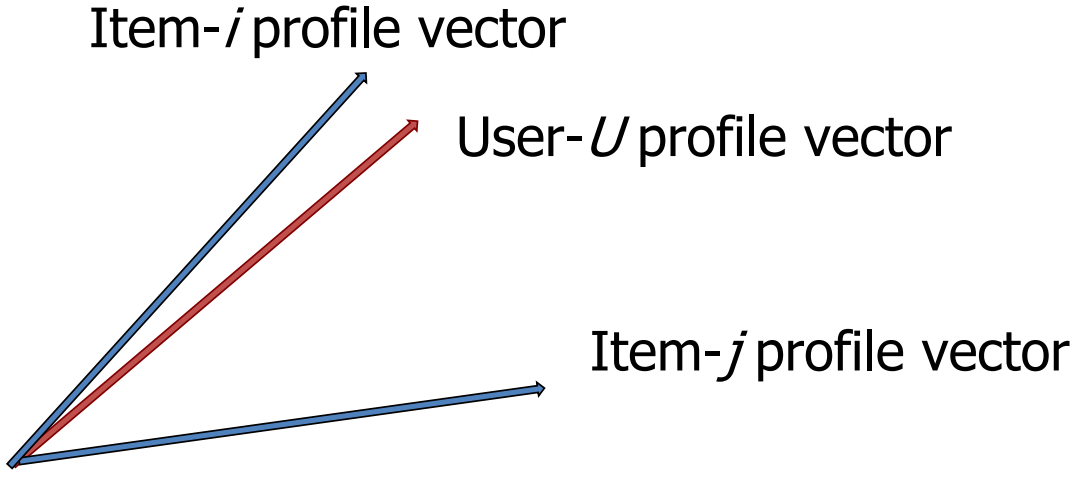


recommend

General Strategy for Content-Based Recommendations

- **Construct item profiles (item-based)**
  - Explicit features in a database, discovering features in documents, Tags
  - **Create vectors representing items**
    - Boolean vectors indicate occurrence of high TF.IDF word
    - Numerical vectors might contain ratings
- **Construct user profiles (user-based)**
  - **Create vectors with same components that describe user's preferences**
- **Recommend items to users based on content**
  - **Calculate cosine distance between item and user vectors**
  - Classification algorithms

Content-Based Recommendation



Item *i* is more similar to user U then *j*
Recommend *i* to *U*

3. Item Profiles

Vector
Content
Image

1)

- For each item, create an **item profile**
- **Profile is a set (vector) of features**
  - **Movies "features":** screenwriter, title, actor, director,…
  - **Text "features":** Set of "important" words in document

- **Example 9.2**
  - Features of movies: a set of (8) actors, and an average rating
    - E.g., see table below, each movie has 5 actors, two in both movies
    - Average ratings 3 and 4 (with unknown scaling factor α)
  - **Must scale non-Boolean components so they are not dominant or irrelevant**

	Actors (features)								Rating
Movies (Items)	0	1	1	0	1	1	0	1	3α
	1	1	0	1	0	1	1	0	4α

Cosine Distance (Section 3.5.4)

- **Cosine distance is used in spaces with dimensions, including Euclidean spaces**
  - Where points are vectors with integer or Boolean components
- Points thought of as directions
- Cosine distance between 2 points is the angle that the vectors to those points make
  - Range from 0 to 180 degrees
- **First compute cosine of the angle** between vectors x and y
- **Then apply arc-cosine function to translate to 0-180 degrees: the cosine distance**

Cosine Similarity
(Used to Calculate Cosine Distance)

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

- **Dot product x.y divided by Euclidean distance of x and y from origin** (Section 3.5.2)
- Dot product of vectors:

$$[x_1, x_2, \ldots, x_n].[y_1, y_2, \ldots, y_n] \text{ is } \sum_{i=1}^{n} x_i y_i.$$

- **Euclidean distance of two vectors x, y**

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Example 3.1.3: Cosine Distance

- x = [1,2,-1] y = [2,1,1] $\cos\theta = \frac{2+2-1}{\sqrt{6}\times\sqrt{6}} = \frac{1}{2} \Rightarrow 60^\circ$
- **Dot product** x.y = 1x2 + 2x1 + (-1)x1 = 3
- **Euclidean distance** of two vectors x, y

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- **Euclidean distance of a vector from the origin: vector for origin is all zeros**
- Distance of x from origin is: $\sqrt{1^2 + 2^2 + (-1)^2} = \sqrt{6}.$
- Distance of y from origin also sqrt(6)
- Cosine of angle from x to y is $3/(\sqrt{6}\sqrt{6})$ or $1/2.$
- **Arccosine** of ½ is 60 degrees
- That is the **cosine distance between x and y**

Back to Item Profiles Example (9.2)

- Features of movies are set of actors and average rating (scaled)

$$\begin{array}{cccccccccc} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 3\alpha \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 4\alpha \end{array}$$

- Dot product of these two item vectors is:

$$[x_1, x_2, \ldots, x_n].[y_1, y_2, \ldots, y_n] \text{ is } \sum_{i=1}^{n} x_i y_i.$$

  - Or $2 + 12\alpha^2$

- Distance from origin using

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- Lengths of vectors (distance from origin) are:

$$\sqrt{5 + 9\alpha^2} \text{ and } \sqrt{5 + 16\alpha^2}$$

- Cosine of angle between vectors is $\dfrac{2 + 12\alpha^2}{\sqrt{25 + 125\alpha^2 + 144\alpha^4}}$
- Scaling factor alpha affects how similar items are

Item Profiles based on Textual Content

- Much research on content-based recommendations focuses on **textual content**
  - Recommend items (web pages, books, movies) based on associated textual content
  - Descriptions, user reviews
- Can treat this as an **Information Retrieval task (IR)**
- **How to identify whether two documents are about similar things?**
- **How to pick important features of documents?**
- Want to **identify the significant words** in documents

TF.IDF: Measure of Word Importance

- Classification of documents as being about similar things starts with finding significant words in those documents
- **Not most frequent words**
  - (the, and, a, …) – called "stop words"
- **Not just rare words either**
- **Want concentration of useful words in just a few documents**
- Usual heuristic from text mining is **TF-IDF:**

  (Term frequency * Inverse Doc Frequency)
- **Words with highest TF.IDF score are often the terms that best characterize the topic of a document**
- When constructing an item profile for Recommender system:
  - **Document … Item**
  - **Term … Feature**

TF-IDF: From Section 1.3.1
(Term frequency * Inverse Document Frequency)

f_{ij} = frequency of term (feature) *i* in document (item) *j*

Term Frequency: $TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$

- Term frequency of term i in document j is **normalized**
  - Divide by maximum occurrences of any term in document j
- Most frequent term has TF=1

n_i = number of docs that mention term *i*

N = total number of docs

Inverse Document Frequency: $IDF_i = \log_2(N/n_i)$

TF-IDF score: $w_{ij} = TF_{ij} \times IDF_i$

Item profile for a document = set of words with highest TF-IDF scores, together with their scores

TF.IDF Example (Example 1.3)

- Repository of 2^{20} = 1,048,576 documents
- Suppose word w appears in 2^{10} = 1024 documents
- **Inverse document frequency:**

  $$IDF_w = \log_2(2^{20}/2^{10}) = \log_2(2^{10}) = 10$$ (logarithm scaled)

- Consider document *j* in which word *w* appears 20 times
  - Assume that this is the maximum number of times any word appears in document j (after eliminating stop words)
  - So $TF_{wj} = 1$
  - **TF.IFD score for w in document j is 1*10=10**
- Consider document *k* where word *w* appears once
  - Maximum number of occurrences of any word in k is 20
  - So $TF_{wk} = 1/20$
  - **TF.IDF score for w in document k is (1/20)*10=1/2**

	j	k
w	10	0.5

(4)

Recommender Systems: Make Recommendations Based on Features of Documents

- **Want to suggest articles, pages, blogs a user might want to see**
- **Hard to classify items by topic** → use words with highest TF.IDF.
- In practice, **try to identify words that characterize the topic of a document**
- **Eliminate stop words:** several hundred most common words
- **For remaining words, calculate the TF.IDF score** for each word in the document
- **The words with the highest TF.IDF scores characterize the document**

doc → set of important words

Represent documents by a set of words

topic

- **Take as features of the document the *n* words with highest TF.IDF scores**
  - Could pick **same *n* for all documents**
  - Or let ***n* be fixed percentage** of words in the document
  - Could also make **all words with TF.IDF scores above a given threshold** are part of feature set
- Documents then represented by set of "important" words
- Expect these words to express subjects or main ideas of documents
- **Then can measure the similarity of two documents using:**
  - **Cosine distance between the sets, treated as vectors (last time)**
  - **Jaccard distance (Ch. 3) between sets of word**

Document Similarity using Cosine Distance

- First compute cosine of the angle between vectors A and B

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

- Then apply arc-cosine function to translate to 0-180 degrees: the cosine distance

Recall: Item Profiles Example (9.2)

- Features of movies are set of actors and average rating (scaled)

$$\begin{array}{cccccccccc} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 3\alpha \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 4\alpha \end{array}$$

- Dot product of these two item vectors is:

$$[x_1, x_2, \ldots, x_n].[y_1, y_2, \ldots, y_n] \text{ is } \sum_{i=1}^{n} x_i y_i.$$

  - Or $2 + 12\alpha^2$

- Distance from origin using

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- Lengths of vectors (distance from origin) are:

$$\sqrt{5 + 9\alpha^2} \text{ and } \sqrt{5 + 16\alpha^2}$$

- Cosine of angle between vectors is $\dfrac{2 + 12\alpha^2}{\sqrt{25 + 125\alpha^2 + 144\alpha^4}}$

Document Similarity using Cosine Distance

- **Think of set of high-TF.IDF words as a vector, with one component for each possible word**
- Vector has 1 if word is in the set for that document, 0 if not
- Between two documents, only a finite number of words among their two sets
- Almost all components are 0; do not affect dot product
- **Dot products** are size of intersection of the two sets of words
- **Lengths of vectors** are square roots of number of words in each set
- Cosine of angle between vectors: dot product divided by product of vector lengths

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Document Similarity: Two Kinds

- **Chapter 3: find nearly-identical (similar) documents**
  - **Lexical similarity:** Documents are similar if they contain large fraction of identical sequences of characters
  - **Shingling:** convert documents to sets
  - **Minhashing:** convert large sets to short signatures, preserving similarity
  - **Locality-sensitive hashing (LSH):** Focus on parts of signatures likely to be from similar documents to **identify candidate pairs**
- **For Recommendation Systems (Chapter 9):**
  - Interested in occurrences of **many important words in both documents** (even if little lexical similarity between documents)

Similar methodology:

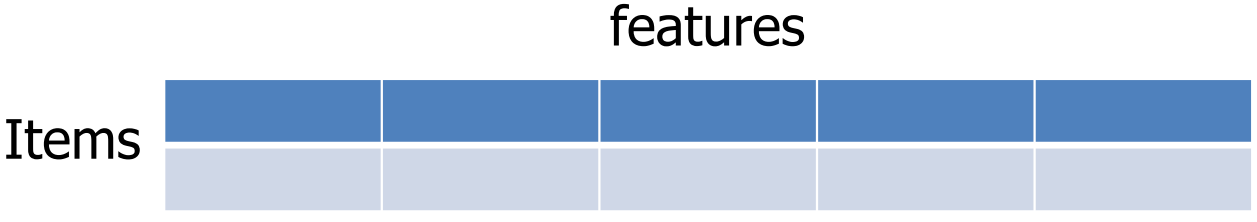
- **High TF.IDF words form a vector**, with a component for each possible word set to 1 or 0 *(analogous to sets of shingles)*
- Based on a **distance measure** (Jaccard or cosine distance)

Another Option to Describe Item Content: Obtaining Item Profile Features from Tagging Systems

- Useful for content-based recommendations for **images**
- E.g., Flickr photosharing, Facebook, Instagram, Snapchat
- Users enter words or phrases that describe items
- GPS information/geofilters: e.g., automatically add location information when a photo is uploaded
- **Can use tags as a recommendation system**
  - E.g., if user retrieves or bookmarks pages with certain tags, recommend other pages with same tags
- Only works if users create tags or allow automatic geotagging

Item Profile (Summary)

- Items = ?
- Features or components of items = ?
- Distance between items = ?



General Strategy for Content-Based Recommendations

- **Construct item profiles**
  - Source we discussed:
    - Explicit features in a database
    - Discovering features in documents
    - Tags
  - **Create vectors representing items**
    - Boolean vectors indicate occurrence of high TF.IDF word
    - Numerical vectors might contain ratings
- **Construct user profiles**
  - **Create vectors with same components that describe user's preferences**
- **Recommend items to users based on content**
  - Calculate cosine distance between item and user vectors
  - Classification algorithms

2.

User Profiles (Examples 9.3 and 9.4)

- **Construct user profiles**
  - Create vectors with same components that describe user's preferences
  - Best estimate regarding which items a user likes is **some aggregation of the profiles of those items**
- **User profile possibilities:**
  - **Boolean utility matrix:** average the components of vectors representing item profiles for the items in which utility matrix has a 1 for that user
    - E.g., 20% of movies that user U likes have actor A (has a 1)
    - User profile for U will have 0.2 in component for actor A
  - **Non-boolean utility matrix: (e.g., ratings)** weight the vectors representing profiles of items by utility (rating) value
    - U gives average rating of 3; rates three movies with actor A: 3, 4, 5
    - Normalize by subtracting user's average rating: new ratings 0, 1, 2
    - Then user profile component for actor A will have value of 1
    - **Negative weights for below-average ratings, positive for above-avg.**

Content-Based Recommendations

- **Prediction Heuristic**
  - Given user profile ***x*** and item profile ***i***
  - **Estimate degree to which a user would prefer an item by computing cosine distance between *x* and *i* vectors**
- **Classification Algorithms**
  - Use **machine learning techniques**
  - Regard given data as a training set
  - For each user, **build a classifier that predicts the rating of all items**
  - Ratings on a scale of 1 to k can be directly mapped to k classes
  - Many different classifiers
    - Naïve Bayes classifier
    - K-nearest neighbor
    - Decision trees
    - Neural networks

Example 9.5

- Building User Profile in Previous Example (9.4):
  - U gives average rating of 3; rates three movies with actor A: 3, 4, 5
  - Normalize by subtracting user's average rating: new ratings 0, 1, 2
  - Negative weights for below-average ratings, positive for above-average
- **Movie with many actors the user likes:**
  - **Cosine of angle will be large positive fraction**
  - **After applying the arccosine, will have a small cosine distance between vectors (angle close to 0 degrees)**
- **Movie with mix of actors the user likes/doesn't like: Cosine of angle will be around 0 (angle close to 90 degrees)**
- **Movie with many actors user doesn't like: Cosine will be a large negative fraction => large cosine distance between vectors (angle close to 180 degrees)**

Thought Exercise

Item Profile

	Actors (features)								Rating
Movie1	0	1	1	0	1	1	0	1	3α
Movie2	1	1	0	1	0	1	1	0	4α

User Profile

User U	.1	.5	.7	.1	.4	.7	.1	.6
User V	.2	.8	.1	.9	.2	.9	.7	.1

Which movie should we recommend for user U and V?

distance(Movie1, UserU) = ?
distance(Movie2, UserU) = ?
distance(Movie1, UserV) = ?
distance(Movie2, UserV) = ?

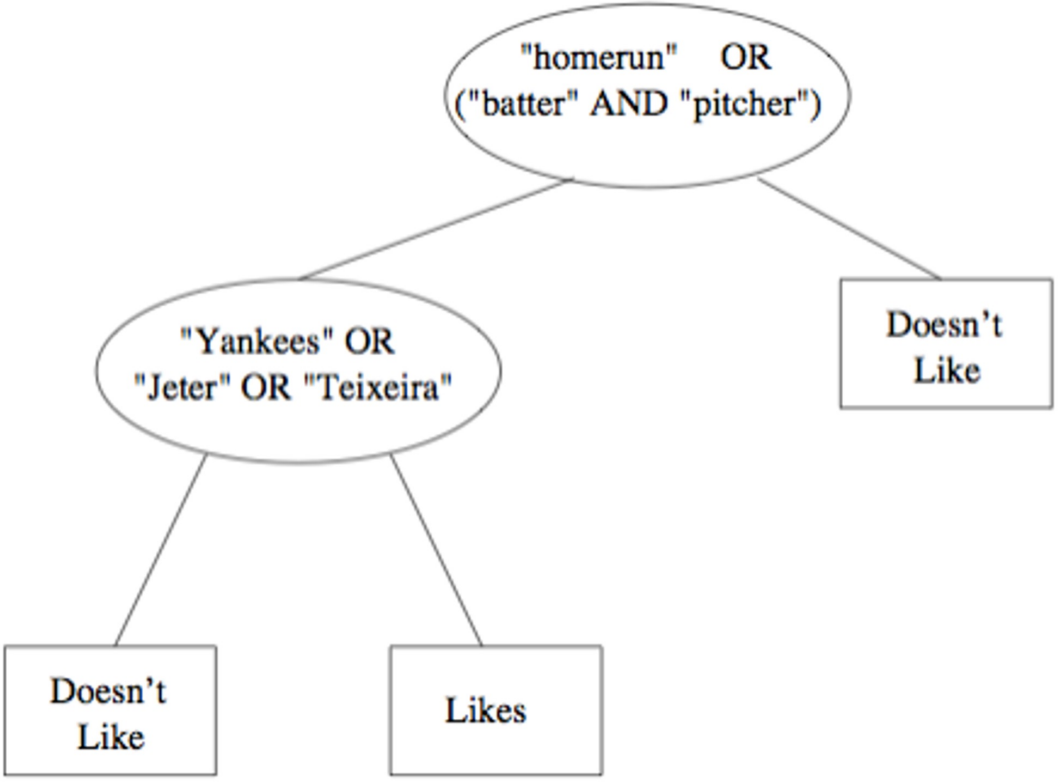
Recommending items based on cosine distance

- Estimate degree to which a user would prefer an item by **computing cosine distance** between x and i vectors
- Scale components with values that are not boolean (e.g., ratings)
- Use Random hyperplanes (RH) and Locality Sensitive Hashing (LSH) techniques to place item profiles (i vectors) in buckets
- **For a given user (x vector), apply RH and LSH techniques: identify in which bucket we look for items that might have a small cosine distance from user**

Example 9.6: Decision Trees

- Items are **news articles**
- **Features are high TF.IDF words (keywords) in these documents**
- Suppose user U likes articles about baseball except articles about New York Yankees
- Row of utility matrix for U has a 1 if U has read the article, blank otherwise
- **Construct decision tree:**
  - select a predicate for each interior node
- **Classify an item:**
  - Start at root and apply predicate to the item
  - If predicate is true, go to left child; if false, go to right child
  - Repeat until a leaf is reached: leaf classified liked or not liked

Example 9.6 (cont.)



Classifiers

- **Classifiers of all types take a long time to construct**
  - E.g., for decision trees: need one tree per user
- Constructing a tree requires looking at all item profiles
- Have to consider many different predicates
- Could involve complex combinations of features
- Typically applied only to small problem sizes

Pros: Content-based Approach

- **+: No need for data on other users**
  - No cold-start (for item) or sparsity problems (i.e., new items can receive recommendations)
- **+: Able to recommend to users with unique tastes**
- **+: Able to recommend new & unpopular items**
  - No first-rater problem (i.e., new products never have been rated, therefore they cannot be recommended)
- **+: Able to provide explanations**
  - Can provide explanations of recommended items by listing content-features that caused an item to be recommended

Cons: Content-based Approach

- –: Finding the appropriate features is hard
  - E.g., images, movies, music
- –: Recommendations for new users
  - How to build a user profile?
- –: Overspecialization
  - Never recommends items outside user's content profile
  - People might have multiple interests
  - Unable to exploit quality judgments of other users (don't use ratings!)

(2)

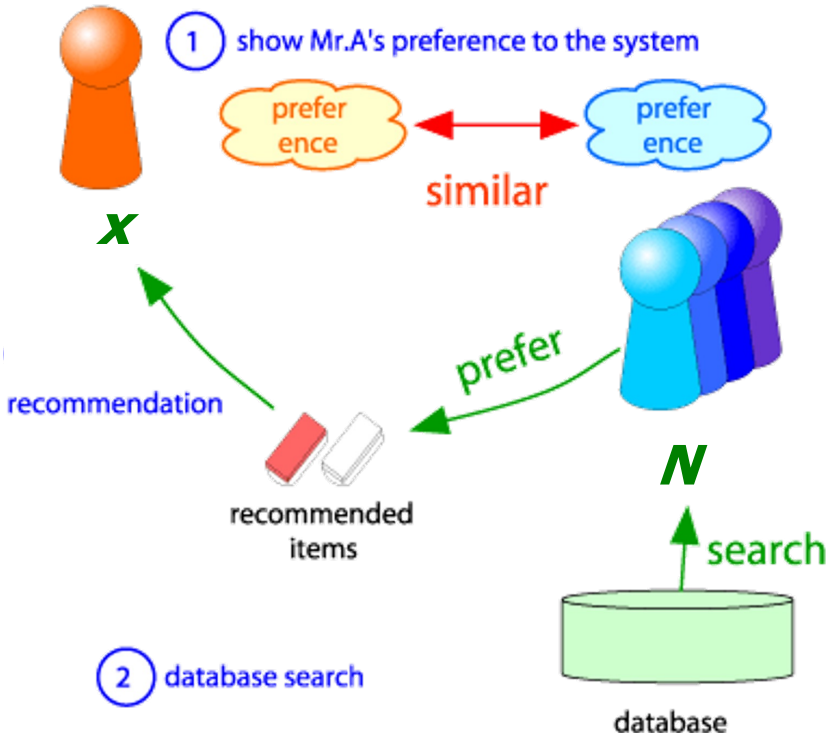
Collaborative Filtering

Harnessing quality judgments of other users or items

- **2) Collaborative filtering (consider other users)**
  - Build a model from
    - a user's past behavior (e.g., items previously purchased or rated), and
    - similar decisions made by other users
  - Use the model to predict items that the user may like
  - Collaborative: suggestions made to a user utilizing information across the entire user base

Collaborative Filtering Example

- **User-based** collaborative filtering
- Consider user ***x***
  - Find set ***N*** of other users whose ratings are "**similar**" to ***x***'s ratings
  - Estimate ***x***'s ratings based on ratings of users in ***N***



Collaborative Filtering: Overview

- CF works by **collecting user feedback**: e.g., **ratings for items**
  - Exploit similarities in rating behavior among users in determining recommendations
- **Two classes of CF algorithms:**

1. **Neighborhood-based or Memory-based approaches**
   - User-based CF
   - Item-based CF
2. **Model-based approaches** fit new user in model
   - Estimate parameters of statistical models for user ratings
   - Latent factor and matrix factorization models

Neighborhood-based / Memory-based Collaborative Filtering

2. USER-BASED CF

Identify User's Neighbors => Recommendation

- Active user: the user we want to make predictions for
- **User-based CF:** A subset of other users (**neighborhood**) is chosen based on their similarity to the active user
- A weighted combination of the ratings of neighbors is used to make predictions for the active user

Steps:

1. Assign a weight to all users w.r.t. **similarity with the active user**
2. **Select k neighbor users that have the highest similarity** with active user (the neighborhood)
3. **Compute a prediction from a weighted combination of the selected neighbors' ratings**

(2) Similarity between users: by what measure?

- **Weight $w_{x,y}$ is measure of similarity between user *x* and active user *y***

Let r_x be the vector of user ***x***'s ratings

Possible similarity metrics:

- **Jaccard similarity**
  - (Intersection / union) of two sets
  - Doesn't use non-boolean values: e.g., ratings
- **Cosine similarity**
  - Treat ratings as vectors to points
  - Cosine similarity between points

r_x, r_y as sets:

$r_x = \{1, 4, 5\}$

$r_y = \{1, 3, 4\}$

r_x, r_y as points:

$r_x = \{1, 0, 0, 1, 3\}$

$r_y = \{1, 0, 2, 2, 0\}$

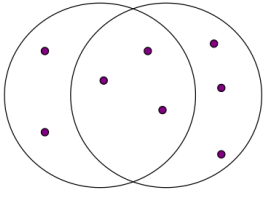
$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

Jaccard Similarity and Distance of Sets

- The ***Jaccard similarity*** of two sets is the size of their intersection divided by the size of their union

$$Sim\ (C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$

- ***Jaccard distance*** = 1 – Jaccard Similarity



3 in intersection
8 in union

Jaccard similarity = 3/8

- **Jaccard distance** = 1 – Jaccard Similarity or 5/8 in this example

How well do these similarity metrics work?

Movies

Users

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

- **Intuitively we want: sim(*A*, *B*) > sim(*A*, *C*) (why?)**
  - For A and B: One movie rated in common with similar ratings
  - For A and C: Two movies rated in common but with dissimilar ratings
- **Jaccard similarity (Example 9.7)**
  - Ignores values (rates) in matrix
  - Only look at which items are rated in matrix
- Intersection / union: sim(A,B) = 1/5, sim(A,C) = 2/4
- 1/5 < 2/4 indicates sim(A,B) < sim(A,C)
- **Not a good similarity metric for ratings data!**

Same matrix with cosine similarity (Example 9.8)

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

- Treat blanks as 0 value: **questionable, since it treats no rating as more similar to disliking a movie than liking it**
- Cosine similarity of A and B is:

$$\frac{4 \times 5}{\sqrt{4^2 + 5^2 + 1^2}\sqrt{5^2 + 5^2 + 4^2}} = 0.380$$

- Cosine similarity of A and C is: 0.322
- 0.380 > 0.322: **Indicates A,B slightly more similar than A,C**
- **in this case, cosine similarity better than Jacquard similarity**

Normalizing ratings (Example 9.9)

- To deal better with non-rated items
- Subtract the average rating of that user from each rating
  - low ratings -> negative numbers; high ratings -> positive numbers

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Avg. Rating
10/3
14/3
11/3
6/2

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	$2/3$			$5/3$	$-7/3$		
B	$1/3$	$1/3$	$-2/3$				
C				$-5/3$	$1/3$	$4/3$	
D		0					0

(V – Avg)

Cosine sim A,B vs. A,C: 0.092 **>** -0.559 (85 degrees vs 124 degrees)
A, C are much further apart than A, B, but neither is close

Most commonly used measure of similarity:

④ Pearson Correlation Coefficient

- **Pearson correlation measures extent to which two variables linearly related**
- For users u, v: Pearson correlation is

average average normalised cosine

$$w_{u,v} = \frac{\sum_{i \in I} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I} (r_{v,i} - \bar{r}_v)^2}}$$

where the $i \in I$ summations are over the items that both the users u and v have rated and $\bar{r}_u$ is the average rating of the co-rated items of the uth user.

- And $r_{u,i}$ is rating of item i by user u

- **Note: When calculating these similarities, average only over the co-rated items** // so be careful

Example: Pearson Correlation Coefficient

$$w_{u,v} = \frac{\sum_{i \in I} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I} (r_{v,i} - \bar{r}_v)^2}}$$

	I_1	I_2	I_3	I_4	
U_1	4	?	5	5	14/3
U_2	4	2	1		
U_3	3		2	4	
U_4	4	4			
U_5	2	1	3	5	10/3 ~~11/4~~

- Want correlation of user 1, user 5: $w_{1,5}$ **Set *I* of co-rated movies = {I_1, I_3, I_4}**
- For user 1: **average rating on co-rated movies** is 14/3; For user 5: 10/3
- Numerator: (4 - 14/3)(2 - 10/3) + (5- 14/3)(3- 10/3) + (5-14/3)(5-10/3)
  = 12/9 = 1.3333
- Denominator: sqrt[(4-14/3)^2 + (5-14/3)^2 + (5-14/3)^2]
  *sqrt[(2-10/3)^2 + (3-10/3)^2 + (5-10/3)^2]
  = 1.76383
- **Pearson correlation $w_{1,5}$= 1.3333 / 1.76383 = 0.756**

Example: Pearson Correlation Coefficient

$$w_{u,v} = \frac{\sum_{i \in I} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I} (r_{v,i} - \bar{r}_v)^2}}$$

	I_1	I_2	I_3	I_4
U_1	4	?	5	5
U_2	4	2	1	
U_3	3		2	4
U_4	4	4		
U_5	2	1	3	5

- Want correlation of user 1, user 4: $w_{1,4}$ **Set *I* of co-rated movies = {?,?}**
- For user 1: **average rating on co-rated movies** is ?; For user 4 is ?
- Numerator:
- Denominator:
- **Pearson correlation $w_{1,4}$ = 0**

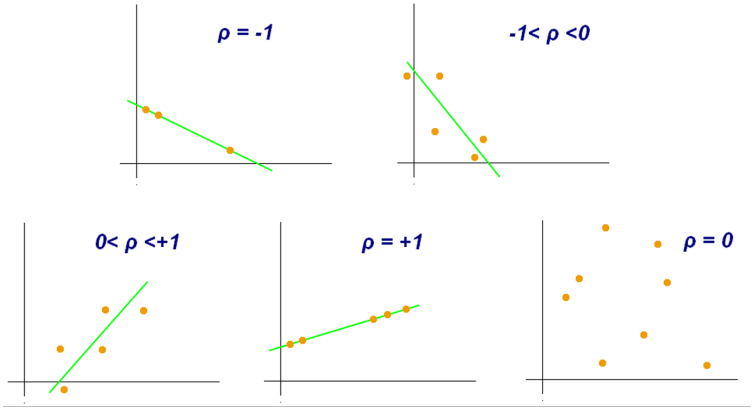
Example: Pearson Correlation Coefficient

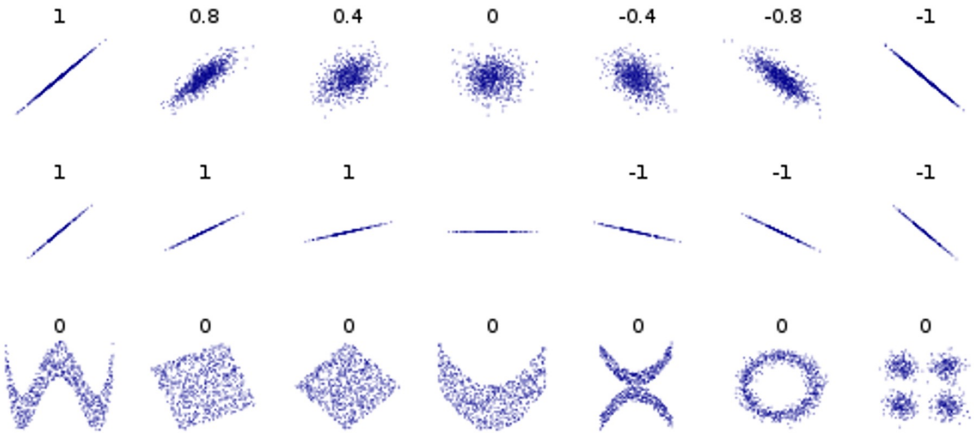
$$w_{u,v} = \frac{\sum_{i \in I} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I} (r_{v,i} - \bar{r}_v)^2}}$$

	I_1	I_2	I_3	I_4	
U_1	4	?	5	5	9/2
U_2	4	2	1		5/2
U_3	3		2	4	
U_4	4	4			
U_5	2	1	3	5	

- Want correlation of user 1, user 2: $w_{1,2}$ **Set *I* of co-rated movies = {I_1, I_3}**
- For user 1: **average rating on co-rated movies** is 9/2; For user 2: 5/2
- Numerator: (4–4.5)(4–2.5) + (5-4.5)(1-2.5) = -0.5(1.5) + 0.5(-1.5) = -1.5
- Denominator: sqrt((4-4.5)^2 + (5-4.5)^2)*sqrt((4-2.5)^2 + (1-2.5)^2)
  = sqrt(0.5)*sqrt(1.5^2*+1.5^2) = sqrt(0.5)sqrt(4.5) = 1.5
- **Pearson correlation $w_{1,2}$ = -1.5 / 1.5 = -1**

Pearson Correlation Examples





Source: https://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient

(3)

Making User-based CF Predictions with Pearson: Weighted Sum of Other Users' Ratings

- Weighted average of their ratings is used to generate predictions
- To make a prediction for an active user *a* on an item i:

Your own opinion

other people's opinion

neighbour

$$P_{a,i} = \bar{r}_a + \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_u) \cdot w_{a,u}}{\sum_{u \in U} |w_{a,u}|}$$

How much you like u

pearson correlation

where $\bar{r}_a$ and $\bar{r}_u$ are the average ratings for the user a and user u on all other rated items, and $w_{a,u}$ is the weight between the user a and user u. The summations are over all the users $u \in U$ who have rated the item i. F

$\bar{r}_a$ & $\bar{r}_u$ → not corated items

- Note: When making predictions, calculate average of ALL rated items for users *a* and *u*
- Summation is over all users who rated item *i*

Continued Example: User-Based CF Prediction with Pearson Correlation Coefficient

	I_1	I_2	I_3	I_4
U_1	4	?	5	5
U_2	4	2	1	
U_3	3		2	4
U_4	4	4		
U_5	2	1	3	5

"average other than I2"

(4+5+5)/3

(4+1)/2

(4)/1

(2+3+5)/3

- **Want to predict rating for user U1 on item I2:** users U2, U4 and U5, who rated I2
- **Similarity of U1 to these users:** $w_{1,5} = 0.756$, $w_{1,4} = 0$, $w_{1,2} = -1$

$$P_{1,2} = \bar{r}_1 + \frac{\sum_u (r_{u,2} - \bar{r}_u) \cdot w_{1,u}}{\sum_u |w_{1,u}|}$$

14/3 = 4.67

$$= \bar{r}_1 + \frac{(r_{2,2} - \bar{r}_2) w_{1,2} + (r_{4,2} - \bar{r}_4) w_{1,4} + (r_{5,2} - \bar{r}_5) w_{1,5}}{|w_{1,2}| + |w_{1,4}| + |w_{1,5}|}$$

5/2=2.5

4/1=4

10/3=3.33

4+5+5 / 3

$$= 4.67 + \frac{(2 - 2.5)(-1) + (4 - 4)0 + (1 - 3.33)0.756}{1 + 0 + 0.756}$$

$$= 3.95.$$

Neighborhood-Based algorithms

- In neighborhood-based CF algorithms, **a subset of nearest neighbors** of the active user are **chosen based on their similarity with active user**
- Use these users for predictions rather than all users who have rated the item

Item-based Collaborative Filtering

- Neighborhood-based CF algorithms do not scale well when applied to millions of users & items
  - Due to computational complexity of search for similar users (possible solutions?)
- **Item-to-item collaborative filtering**
  - Rather than matching similar users
  - **Match user's rated items to similar items**
- In practice, often leads to faster online systems and better recommendations
- **Similarities between pairs of items *i* and *j* are computed off-line**
- Predict rating of user "***a***" on item "***i***" with a simple weighted average

Pearson Correlation between items *i, j*

For the item-based algorithm, denote the set of users $u \in U$ who rated both items i and j, then the *Pearson Correlation* will be

$$w_{i,j} = \frac{\sum_{u \in U} \left(r_{u,i} - \overline{r}_i\right)\left(r_{u,j} - \overline{r}_j\right)}{\sqrt{\sum_{u \in U} \left(r_{u,i} - \overline{r}_i\right)^2} \sqrt{\sum_{u \in U} \left(r_{u,j} - \overline{r}_j\right)^2}}, \quad (2)$$

where $r_{u,i}$ is the rating of user u on item i, $\overline{r}_i$ is the average rating of the ith item by those users, see Figure 2 [40].

- **Note: Sum over set of users *U* who rated both items *i, j***
- $r_{u,i}$ is rating of user *u* on item *i*
- $\overline{r_i}$ is average rating of i^{th} item by those users

Pearson Correlation between items i, j (Cont'd)

Items

Users

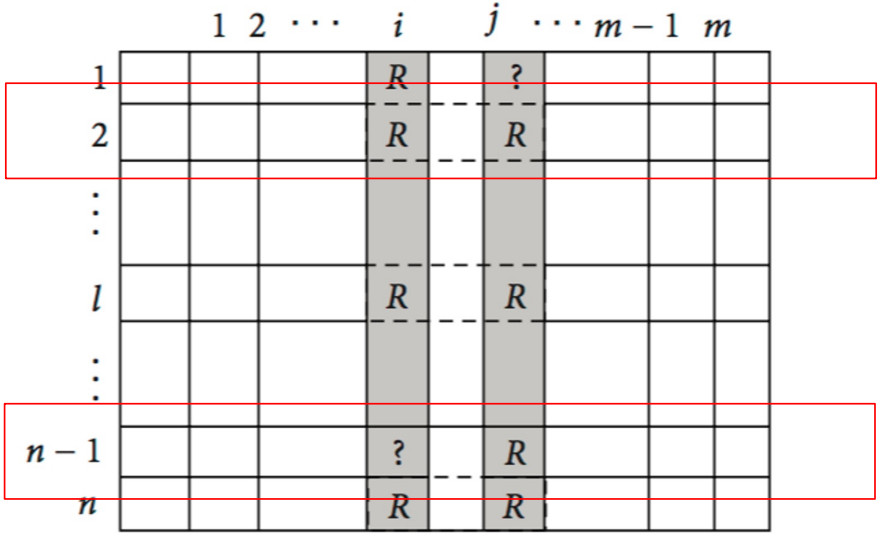


FIGURE 2: item-based similarity ($w_{i,j}$) calculation based on the co-rated items i and j from users 2, l and n.

Source: Su, X., & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Advances in artificial intelligence*, *2009*, 4.

Make Item-Based Predictions Using a Simple Weighted Average

- In order to predict rating for user *u* on item *i*, *do*
- $w_{i,n}$ is the (similarity) weight between item *i* and item *n*
- $r_{u,n}$ is rating for user *u* on item *n*
- Summation over **neighborhood set *N* of items** rated by *u* that are most similar to *i*

item rated

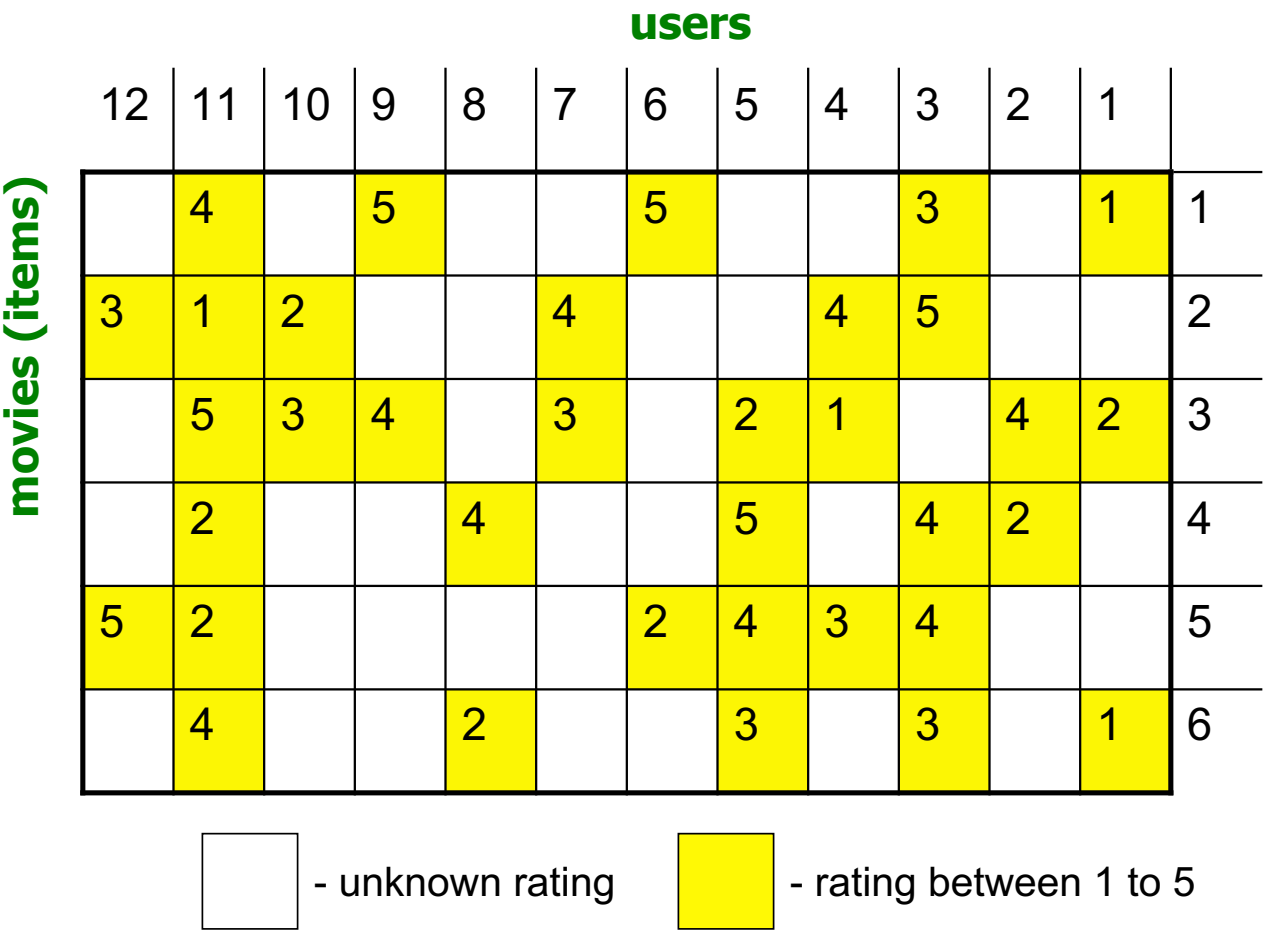
User's opinion on the item *n*

$$P_{u,i} = \frac{\sum_{n \in N} r_{u,n} w_{i,n}}{\sum_{n \in N} |w_{i,n}|}$$

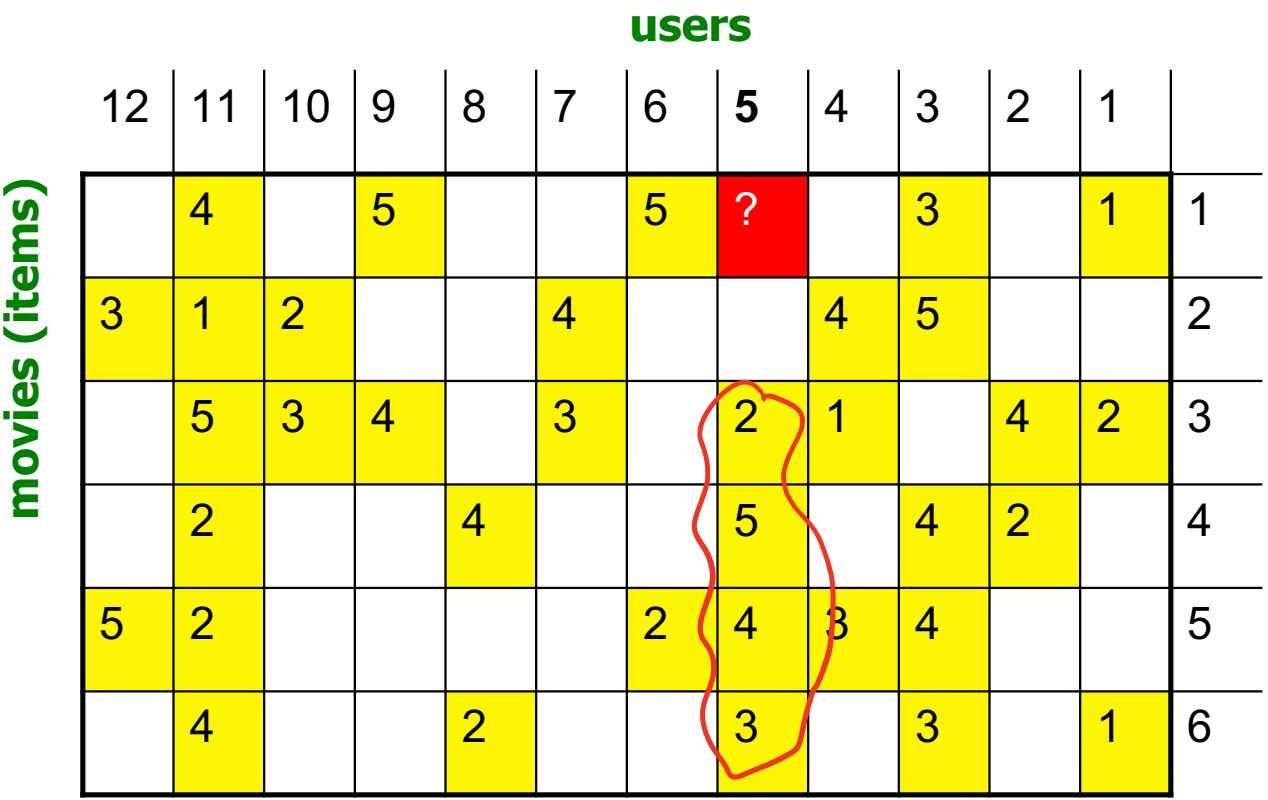
How similar *is n* to *i*

where the summations are over all other rated items $n \in N$ for user u, $w_{i,n}$ is the weight between items i and n, $r_{u,n}$ is the rating for user u on item n.

eg1: Item-Item CF (|N|=2)



Item-Item CF (|N|=2)



$$w_{i,j} = \frac{\sum_{u \in U}(r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum(r_{u,i} - \bar{r}_i)^2}\sqrt{\sum(r_{u,j} - \bar{r}_j)^2}}$$

$$= \frac{(4 - \frac{7}{2})(2 - 3) + (3 - \frac{7}{2})(4 - 3)}{\sqrt{[(4 - \frac{7}{2})^2 + (3 - \frac{7}{2})^2][(2-3)^2 + (4-3)^2]}}$$

$$w_{1,4} = \frac{(r_{1,1} - \bar{r}_1)(r_{1,3} - \bar{r}_3) + (r_{3,1} - \bar{r}_1)(r_{3,3} - \bar{r}_3)}{\sqrt{[(\quad)^2 + (\quad)^2][(\quad)^2 + (\quad)^2]}}$$

users

movies (items)

12	11	10	9	8	7	6	**5**	4	3	2	1	
	4		5			5	?		3		1	1
3	1	2			4			4	5			2
	5	3	4		3		2	1		4	2	**3**
	2			4			5		4	2		4
5	2					2	4	3	4			5
	4			2			3		3		1	**6**

Similarity (made up for example): $w_{1,j}$

1.00

-0.18

0.41

-0.10

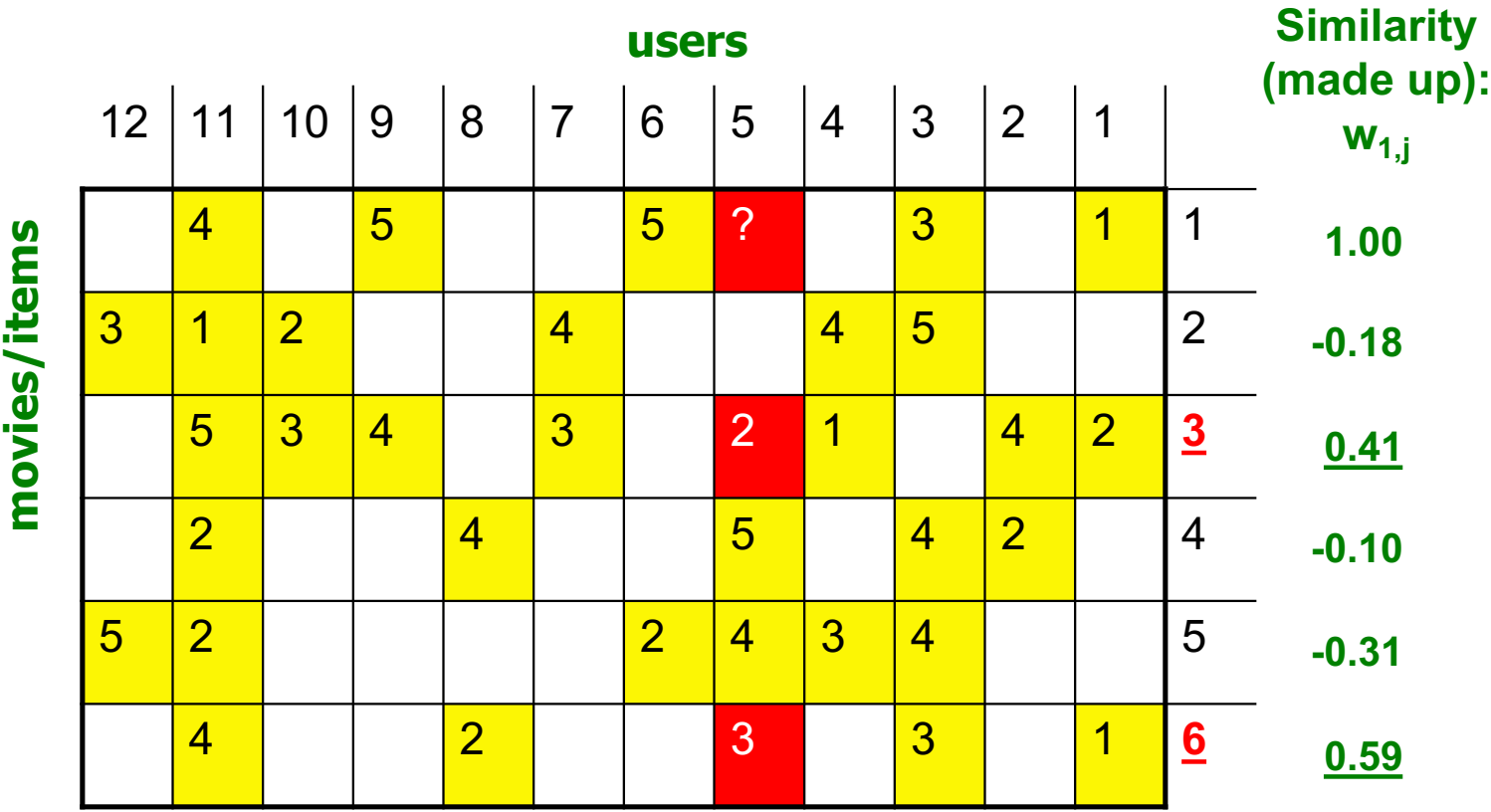
-0.31

0.59

Neighbor selection: Identify movies **most similar to movie 1** and **rated by user 5**

Neighborhood size is 2: pick movies 3 and 6

Item-Item CF (|N|=2)



Similarity weights between items:

$w_{1,3}=0.41$, $w_{1,6}=0.59$

Item-Item CF (|N|=2)

users

movies

12	11	10	9	8	7	6	5	4	3	2	1		Similarity: $w_{1,j}$
	4		5			5	2.6		3		1	1	1.00
3	1	2			4			4	5			2	-0.18
	5	3	4		3		2	1		4	2	3	0.41
	2			4			5		4	2		4	-0.10
5	2					2	4	3	4			5	-0.31
	4			2			3		3		1	6	0.59

Predict by taking weighted average:

$P_{1.5}$ = (0.41*2 + 0.59*3) / (0.41+0.59) = 2.6

$$P_{u,i} = \frac{\sum_{n \in N} r_{u,n} w_{i,n}}{\sum_{n \in N} | w_{i,n} |}$$

Example: Item-to-Item Collaborative Filtering with Pearson Similarity

	I1	I2	I3	I4
U1	2	1		3
U2	3	?	5	2
U3		4	2	3
U4	5	3	1	

$$w_{i,j} = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_i)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \bar{r}_j)^2}}$$

- What does set U represent?
  - *Set of users U who rated both items i, j*
- What are the members of the set U for $w_{1,2}$?
  - *U1 and U4 rated items I1 and I2*
- When calculating average ratings for item i, which ratings do we use? All ratings or just for co-rated items?
  - *We will show examples of* co-rated items
  - *We can use all ratings as well: based on all ratings:* *avg(r_1) = (2+3+5)/3, avg(r_2) = (1+4+3)/3*

① user-based:

Example: Item-to-Item Collaborative Filtering with Pearson Similarity

	I1	I2	I3	I4
U1	2	1		3
U2	3	?	5	2
U3		4	2	3
U4	5	3	1	

$$w_{i,j} = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_i)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \bar{r}_j)^2}}$$

all rated

- *Based on all ratings: average ratings for items I1, I2: $\bar{r}_1$ = 10/3, $\bar{r}_2$ = 8/3*
- Pearson correlation for $w_{1,2}$. *Similarity between items 1 and 2*
- *Set U includes U1 and U4*
- $w_{1,2}$ = (($r_{U1,I1}$ – 10/3)($r_{U1,I2}$ – 8/3) + ($r_{U4,I1}$ – 10/3)($r_{U4,I2}$ – 8/3)) /
  (sqrt(($r_{U1,I1}$ – 10/3) ^ 2 + ($r_{U4,I1}$ – 10/3)^2) * sqrt(($r_{U1,I2}$ – 8/3)^2) + ($r_{U4,I2}$ – 8/3)^2))
- $w_{1,2}$ = [(2-10/3)(1-8/3) + (5-10/3)(3-8/3)] /
  [sqrt((2-10/3)^2 + (5-10/3)^2) * sqrt((1-8/3)^2 + (3-8/3)^2)]
  = 2.778/3.628 = 0.765

Example: Item-to-Item Collaborative Filtering with Pearson Similarity

	I1	I2	I3	I4
U1	2	1		3
U2	3	?	5	2
U3		4	2	3
U4	5	3	1	

$$w_{i,j} = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_i)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \bar{r}_j)^2}}$$

co-rated items only

- *Based on co-ratings: average ratings for items I1, I2: $\bar{r}_1$ = 7/2, $\bar{r}_2$ = 4/2*
- Pearson correlation for $w_{1,2}$. *Similarity between items 1 and 2*
- *Set U includes U1 and U4*
- $w_{1,2}$ = (($r_{U1,I1}$ – 7/2)($r_{U1,I2}$ – 4/2) + ($r_{U4,I1}$ – 7/2)($r_{U4,I2}$ – 4/2)) /
  (sqrt(($r_{U1,I1}$ – 7/2) ^ 2 + ($r_{U4,I1}$ – 7/2)^2) * sqrt(($r_{U1,I2}$ – 4/2)^2) + ($r_{U4,I2}$ – 4/2)^2))
- $w_{1,2}$ = [(2-7/2)(1-4/2) + (5-7/2)(3-4/2)] /
  [sqrt((2-7/2)^2 + (5-7/2)^2) * sqrt((1-4/2)^2 + (3-4/2)^2)]
  =3/3=1

Item-Based Prediction

	I1	I2	I3	I4
U1	2	1		3
U2	3	?	5	2
U3		4	2	3
U4	5	3	1	

$$P_{u,i} = \frac{\sum_{n \in N} r_{u,n} w_{i,n}}{\sum_{n \in N} | w_{i,n} |}$$

- If we have the following item similarities:

  $w_{2,1} = 0.5,\ w_{2,3} = 0.2,\ w_{2,4} = 0.3$

- Which items are in the neighborhood N for item 2 if |N| = 2?
  - Items I1 and I4
- **Predict the rating of user U2 on I2:** user = 2, item = 2, |N| = items 1,4

  $P_{2,2} = [(r_{2,1} * w_{2,1}) + (r_{2,4} * w_{2,4})] / [0.5 + 0.3]$

  $= [3*0.5 + 2*0.3] / 0.8 = 2.625$

Extensions to Memory-Based Algorithms

- A variety of approaches/extensions have been studied to improve the performance of CF predictions
- Typically involve **modifying the similarity weights** or the **ratings** used in predictions or **guessing missing ratings**

- **User-based CF: (User's Neighbors => recommend)**

$$w_{u,v} = \frac{\sum_{i \in I} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I} (r_{v,i} - \bar{r}_v)^2}} \qquad P_{a,i} = \bar{r}_a + \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_u) \cdot w_{a,u}}{\sum_{u \in U} |w_{a,u}|}$$

- **Item-Based CF: (Item's Neighbors => recommend)**

$$w_{i,j} = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_i)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \bar{r}_j)^2}} \qquad P_{u,i} = \frac{\sum_{n \in N} r_{u,n} w_{i,n}}{\sum_{n \in N} |w_{i,n}|}$$

Extensions to Memory-Based Algorithms: Default Voting

- In many collaborative filters, **pairwise similarity is computed only from the ratings in the intersection of the items both users have rated ("co-rated items")**
  - **Not reliable when there are too few votes** to generate similarity values (U is small)
  - Focusing on co-rated items ("intersection set similarity") also **neglects global rating behavior reflected in a user's entire rating history**
- Assuming some default voting values for the missing ratings: **can improve CF prediction performance**

Extensions to Memory-Based Algorithms: Default Voting (cont.)

Approaches to default voting values:

- Herlocker et al. accounts for small intersection sets (small number of co-rated items) by **reducing the weight of users that have fewer than 50 items in common**

$$P_{u,i} = \frac{\sum_{n \in N} r_{u,n} w_{i,n}}{\sum_{n \in N} | w_{i,n} |}$$

- Chee et al. **use average of the clique (small group of co-rated items) as a default voting** to extend a user's rating history
- Breese et al. **use a neutral or somewhat negative preference for the unobserved ratings** and then computes similarity between users on the resulting ratings data.

Extensions to Memory-Based Algorithms: Inverse User Frequency

- **Universally liked items are not as useful in capturing similarity as less common items**
- **Inverse frequency**
  - $f_j = \log (n/n_j)$
  - n_j **is number of users who have rated item** j
  - n **is total number of users**
- If everyone has rated item j, then f_j is zero
  - (Note: looks a lot like Inverse Document Frequency (IDF))
- **Approach: transform the ratings**
  - For vector similarity-based CF: **new rating = original rating multiplied by** f_j

$$P_{a,i} = \bar{r}_a + \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_u) \cdot w_{a,u}}{\sum_{u \in U} | w_{a,u} |}$$

  - **For very popular items, ratings** $r_{u,i}$ **will be greatly reduced**
  - **Less popular items will have greater effect on prediction**

Extensions to Memory-Based Algorithms: (3) Case Amplification

- **Transform applied to weights used in CF prediction**
- Emphasizes high weights and punishes low weights

$$w'_{i,j} = w_{i,j} \cdot |w_{i,j}|^{\rho-1}, \quad (8)$$

where ρ is the *case amplification* power, $\rho \geq 1$, and a typical choice of ρ is 2.5 [65].

- **Reduces noise in the data**
- **Favors high weights**

$$P_{u,i} = \frac{\sum_{n \in N} r_{u,n} w_{i,n}}{\sum_{n \in N} |w_{i,n}|}$$

- Small values raised to a power become negligible
- Example:
  - For $w_{i,j} = 0.9$, weight it remains high ($0.9^{2.5} \approx 0.8$)
  - For $w_{i,j} = 0.1$, weight becomes negligible ($0.1^{2.5} \approx 0.003$)

Extensions to Memory-Based Algorithms: Imputation-Boosted CF

- **When the rating data for CF tasks are extremely sparse: hard to produce accurate predictions using the Pearson correlation-based CF**
- Su et al. proposed imputation-boosted collaborative filtering (IBCF)
- **First uses an imputation technique to fill in missing data**
- **Then use traditional Pearson correlation-based CF algorithm** on this completed data to predict a user rating for a specified item
  - **Example imputation techniques:** mean imputation, linear regression imputation, predictive mean matching imputation, Bayesian multiple imputation, and machine learning classifiers (including naıve Bayes, SVM, neural network, decision tree, lazy Bayesian rules)

Extensions to Memory-Based Algorithms: Imputation-Boosted CF (Cont'd)

Simple mean imputation

The data on the left below has one missing observation on variable 2, unit 10.

We replace this with the arithmetic average of the observed data *for that variable*. This value is shown in red in the table below.

Unit	Variables 1	2
1	3.4	5.67
2	3.9	4.81
3	2.6	4.93
4	1.9	6.21
5	2.2	6.83
6	3.3	5.61
7	1.7	5.45
8	2.4	4.94
9	2.8	5.73
10	3.6	5.58

- This approach is clearly inappropriate for categorical variables.
- It does not lead to proper estimates of measures of association or regression coefficients. Rather, associations tend to be diluted.
- In addition, variances will be wrongly estimated (typically under estimated) if the imputed values are treated as real. Thus inferences will be wrong too.

Imputation Example

Source:
http://missingdata.lshtm.ac.uk/index.php?option=com_content&view=article&id=68:simple-mean-imputation&catid=39:simple-ad-hoc-methods-for-coping-with-missing-data&Itemid=96