

ACA ML 2018

Workshop on Recommender Systems

Part 1

About me

- Evgeny Frolov
evgeny.frolov@skoltech.ru
- Received PhD in SkolTech last week; you can find materials at
<https://www.skoltech.ru/en/2018/09/phd-thesis-defense-evgeny-frolov/>
- Working in the group of prof. Ivan Oseledets + Sberbank AI Laboratory
- Previously graduated from MSU, Physics Department
- Approx. 5 years experience in IT industry

What are your
expectations?

What this course is about

Collection of hints for conducting research and trying ideas.

Practical recipes for using popular recommendation algorithms.

Basic algorithms' implementation and their quality evaluation.

Learning by doing!

Major focus:

Avoid answering wrong questions and solving wrong problems.

*"He points out that one of the really tough things is figuring out what questions to ask...
Once you figure out the question, then the answer is relatively easy."*
Elon Mask's reflections on the "The Hitchhiker's Guide to the Galaxy", by Douglas Adams.

So what is the main question?

- user satisfaction
- company revenue
- does user satisfaction correlate with revenue?
- what is mathematical formulation?
- how to measure it?

there're various metrics, but for many of them there's no convenient math formulation

From practical experience:

even when objective defined, there's still no guarantee that the result will correspond to real RS performance

#recommender_systems

Jump • Oct 31st, 2016

2:19 PM **natekin** так и с юзером, не поняв что у него в голове,

2:24 PM **hushpar** У нас **рандом** один раз показал результаты
лучше, чем у хороших моделей



What this course is NOT about

- Graphs and decision trees
- Neural Networks and Deep learning
- Building complex RS models and ensembles
- Distributed systems, map-reduce
- Databases, scalability
- Hypothesis verification and user studies
- Hand-crafted, association mining or business rule based approaches

Course structure

Course repo: https://github.com/Evfro/acaml2018_recsys

- Part 1
 - Introduction, history, key challenges and trends in recommender systems.
 - Evaluation principles of recommender systems.
 - Recommender systems taxonomy. Content-based and collaborative filtering models.
 - Non-personalized CF models and baseline predictors. Memory-based techniques.
 - Working with *Polara* framework.
- Part 2
 - Collaborative filtering: model-based techniques. Matrix factorization methods in collaborative filtering.
 - Factorization machines and FeatureSVD. Hybrid models.
 - *Practical session*: building and evaluating competitive recommendation model. Individual work.
- Part 3
 - Advanced methods in recommender systems. Context-awareness and tensor factorization.
 - Answering your questions, exploring ideas.
- Practical session
 - Building and evaluating competitive recommendation model. Team contest with public leaderboard.

Helpful resources

Books

- **Recommender Systems Handbook, 2015**, 2nd edition; F. Ricci, L. Rokach, B. Shapira
- **Recommender Systems. The Textbook, 2016**; Charu C. Aggarwal
- **Recommender Systems: An Introduction, 2010**; D.Jannach, M.Zanker, A.Felfernig, G.Friedrich
- **Statistical Methods for Recommender Systems, 2016**; Deepak K. Agarwal, Bee-Chung Chen
- **Collaborative Recommendations: Algorithms, Practical Challenges and Applications**; S. Berkovsky, I. Cantador and D. Tikk; *expected May 2019*.

Conferences

ACM RecSys

UMAP

WWW

KDD

WSDM

Competitions

RecSys Challenge

CIKM challenge

Kaggle

Tip: Want to find a job? Attend RecSys conferences!

Helpful resources

Courses

- <https://www.coursera.org/learn/recommender-systems/home/welcome>
From pioneers of RecSys field
- Week 3 of “Big Data Applications: Machine Learning at Scale” course in Big Data for Data Engineers Specialization, <https://www.coursera.org/specializations/big-data-engineering>
- Machine Learning: Recommender Systems & Dimensionality Reduction
<https://www.coursera.org/learn/ml-recommenders> (Amazon Professors)
- Mining Massive Datasets, Chapter on Recommender Systems, Stanford University
<http://www.mmds.org>

Video tutorials

- **Machine Learning Summer School 2014**
https://www.youtube.com/playlist?list=PLZSO_6-bSqHQCIYxE3ycGLXHMjK3XV7Iz
Lectures from Xavier Amatriain and Deepak Agarwal
- **Introduction to Machine Learning 10-701 CMU 2015**, Alex Smola
<https://www.youtube.com/watch?v=gCaOa3W9kM0>

Other resources

- RecSys wiki: <http://recsyswiki.com> (currently down)
- Blog: **A Practical Guide to Building Recommender Systems**
<https://buildingrecommenders.wordpress.com>
- OpenDataScience Slack, #recommender_systems channel (mostly Russian language)

Libraries and Frameworks

- Frameworks

- Polara (*Disclaimer: I'm the author*)
<https://github.com/evfro/polara>
- MyMediaLite
<http://www.mymedialite.net>
- Collaborative Filtering – Apache Spark
<http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html>
(Neighborhood models and MF)
- Surprise
<https://github.com/NicolasHug/Surprise>
- Turi Create (ex GraphLab Create)
<https://apple.github.io/turicreate/docs/userguide/recommender/>

- Useful libraries

- Collaborative Filtering for Implicit Feedback Datasets
<https://github.com/benfred/implicit> (the fastest)
<https://github.com/quora/qmf> (by Quora)
<https://github.com/MrChrisJohnson/logistic-mf> (as in Spotify)
- Factorization Machines
<https://github.com/srendle/libfm>

- Other libraries

- Neural Networks
<https://github.com/Netflix/vectorflow> (by Netflix)
<https://github.com/amzn/amazon-dsstne> (Amazon)
<https://github.com/maciejkula/spotlight>
<https://github.com/MrChrisJohnson/deep-mf>
<https://github.com/songgc/TF-recomm>
- Bilinear models
<https://github.com/lyst/lightfm/>
<http://www.recsyswiki.com/wiki/SVDFeature>
- Many latent factor models
<https://github.com/zhangsi/CisRec>
- Simple content-based recommendation engine
<https://github.com/groveco/content-engine>
- Logistic Matrix Factorization
<https://github.com/MrChrisJohnson/implicit-mf>
- Hermes (Supports Spark)
<https://github.com/Lab41/hermes>

What is a recommender system?



Examples:

- Amazon
- Netflix
- Pandora
- Spotify
- etc.

Many different areas: e-commerce, news, tourism, entertainment, education...

Goal: predict user preferences given some prior information on user behavior.

Role of personalized services



+\$2.93 billion to revenue after integration of recommendations¹



80% of what people watch comes from recommendations; results in \$1 billion savings²



personalized "Just For You" listings:
"want to be every user's personalized travel guide"³



"younger travelers ... prefer hotel searches closely tailored to their profiles", CEO Darren Huston

¹ <http://fortune.com/2012/07/30/amazons-recommendation-secret/>

² <http://dl.acm.org/citation.cfm?id=2843948>

³ TripAdvisor's annual report, april 2015

Demo

Simple recommendation engine in 3 lines of python code.

Netflix prize story

October 2, 2006 - June 26, 2009



Contest: Given a database of movies rated by users, beat Netflix's recsys by at least 10%

Award: \$1,000,000



Key to success: ensemble of models.

Actual solution was never implemented!

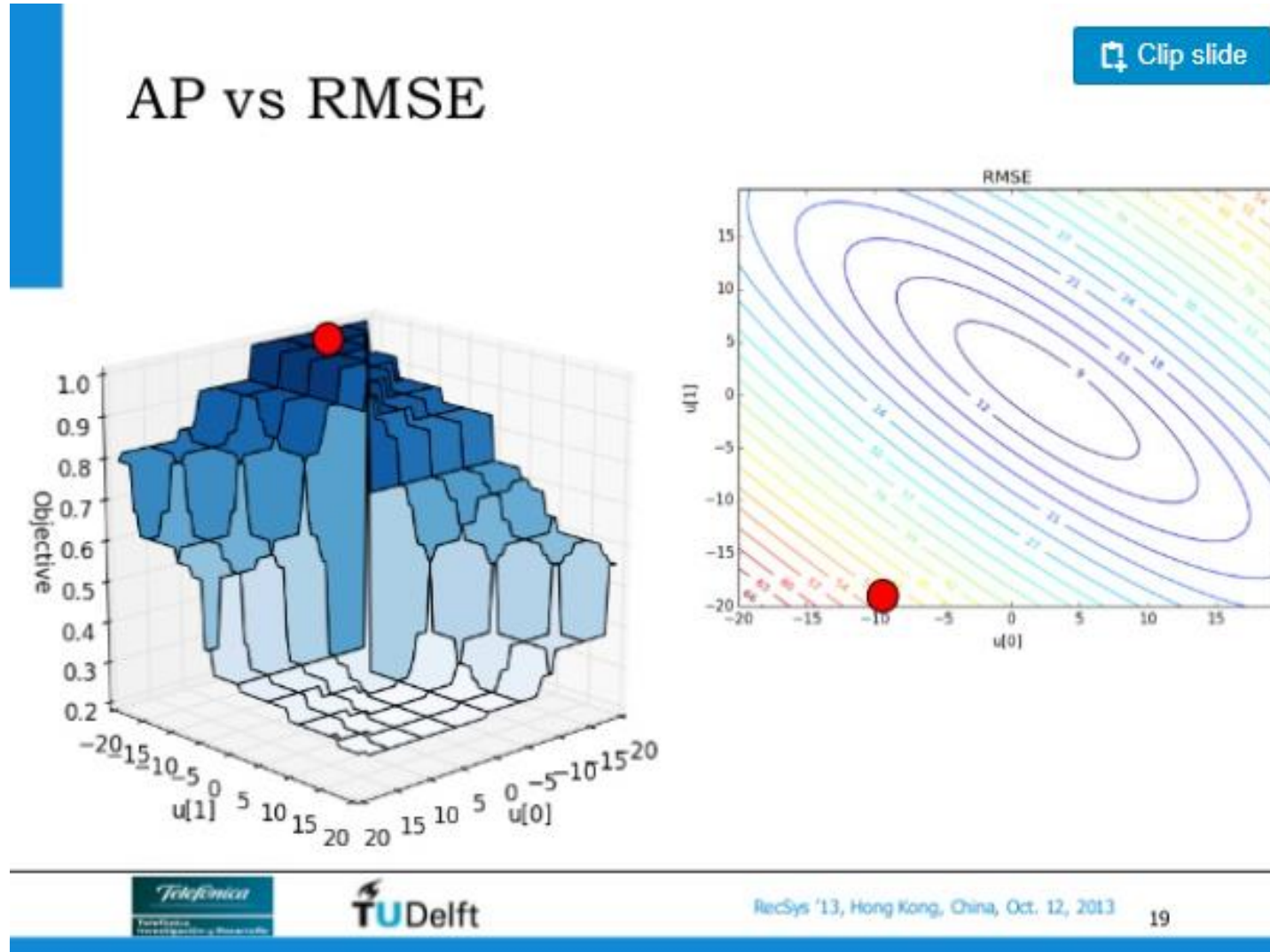
<https://www.techdirt.com/blog/innovation/articles/20120409/03412518422/why-netflix-never-implemented-algorithm-that-won-netflix-1-million-challenge.shtml>

But latent factors models based on **matrix factorization** gained popularity.

Issues:

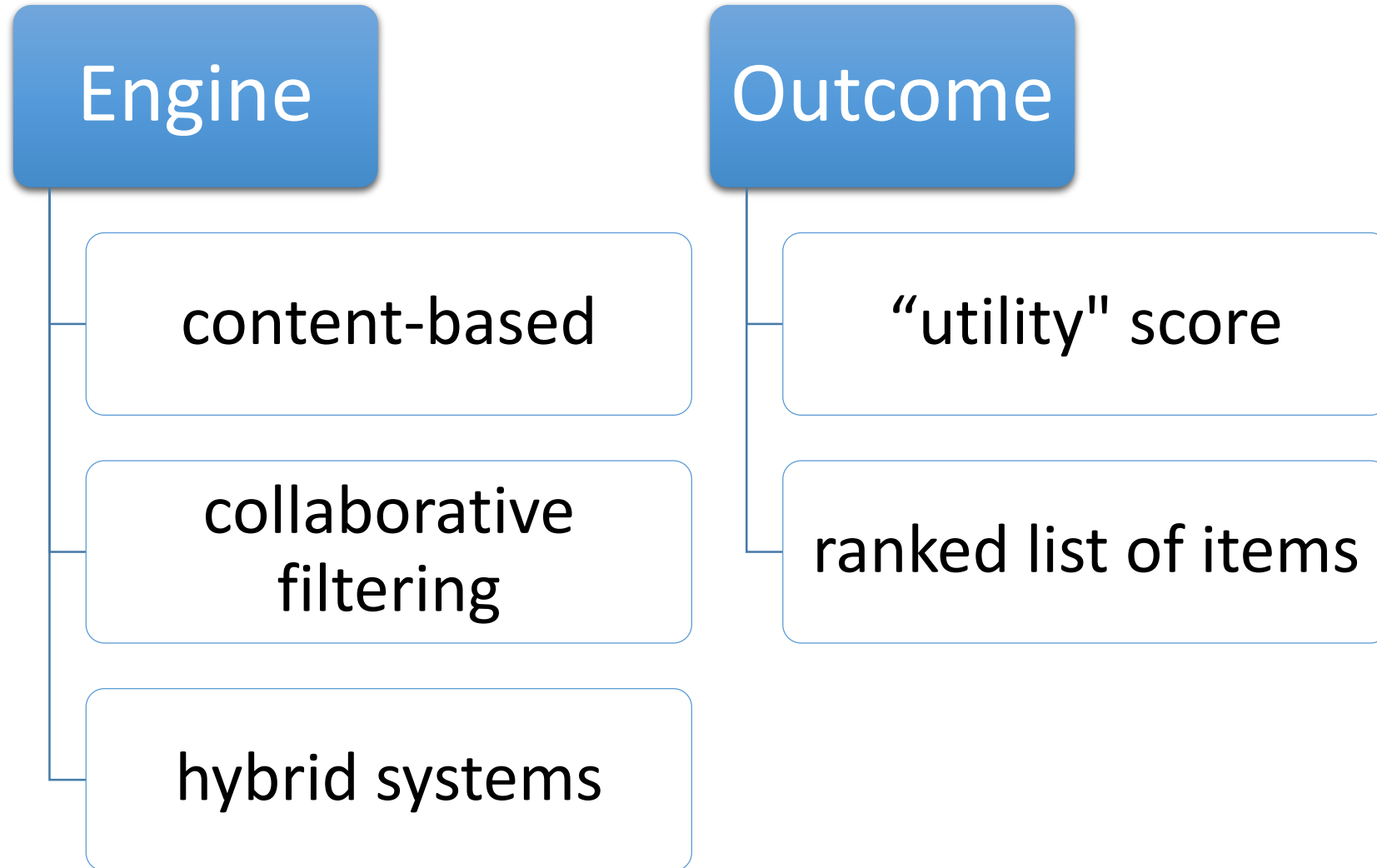
- treated as a pure matrix completion problem
- evaluation metric: RMSE (Precision, Recall, MAP, NDCG, etc. are more adequate)
- proposed matrix factorization models are not better than pure SVD in many practical cases

Error minimization issue



Short break

Recommender systems internals

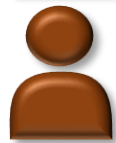


Content-based approach

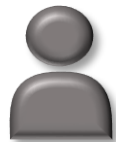
			
Good for gaming	+		+
Good for movies	+	+	+
Good for TV shows	+		
Blue-ray support		+	+



Gamer



Cinema fan



"Family guy"

How to generate recommendations:

item attributes to compute items similarity

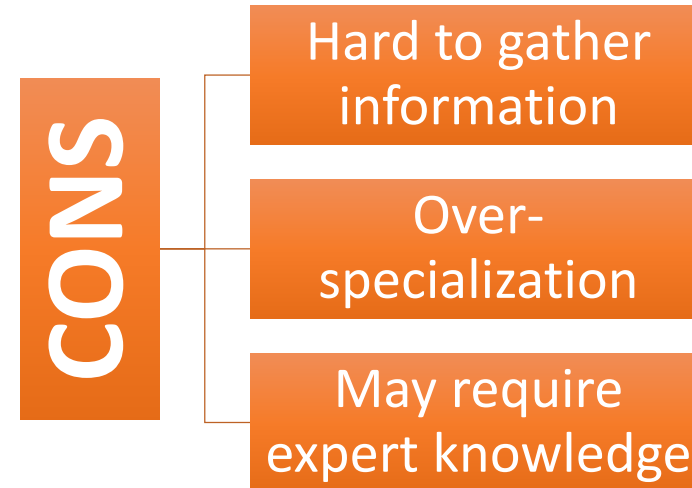
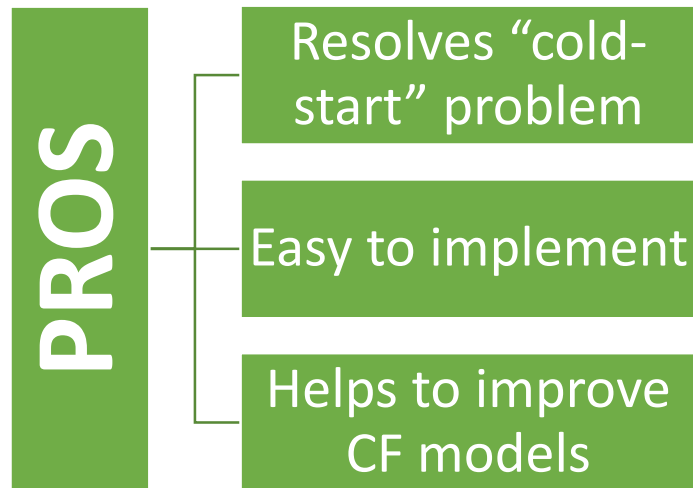
user profile information to match items

previous user actions to match items

Content-based approach

Can use various similarity metrics:

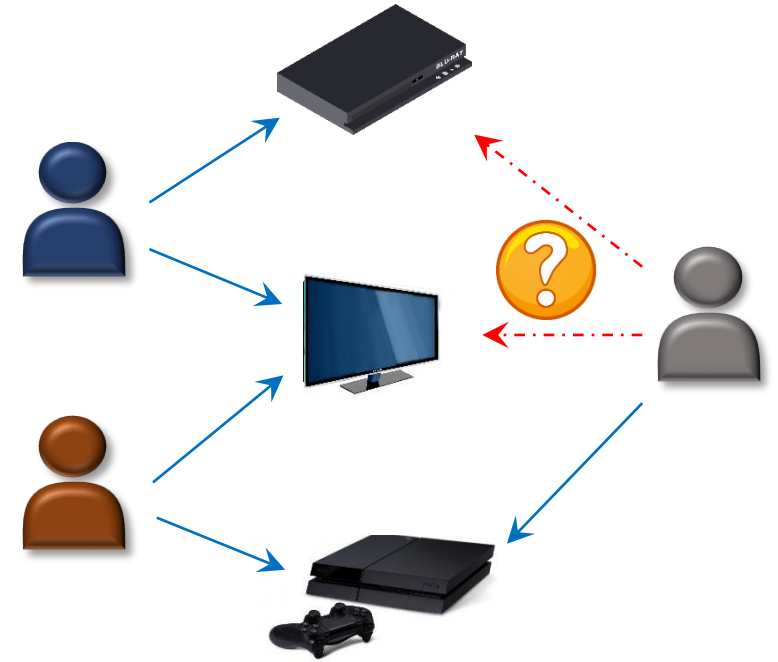
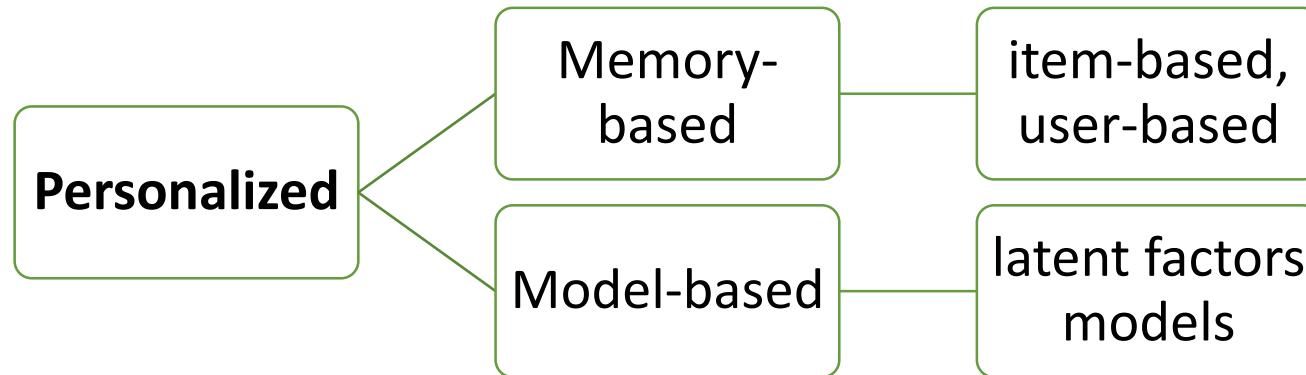
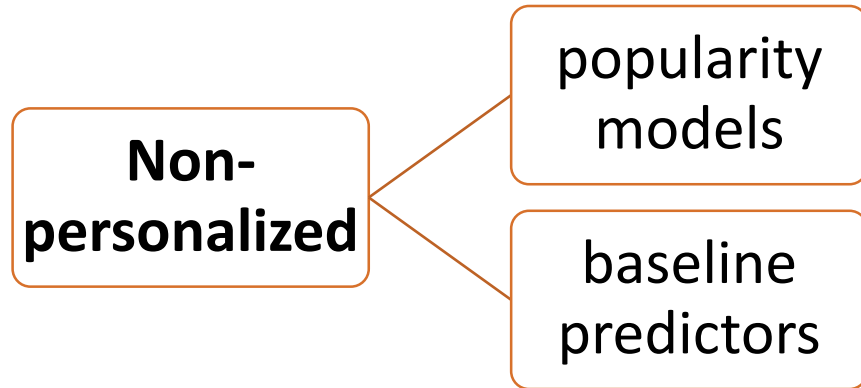
- Cosine similarity
- Jaccard index
- Pearson correlation
- ...



That's where **deep learning** can really help: **features embedding**.

Better in hybrid approaches.

Collaborative Filtering



General workflow

Goal: predict user preferences based on prior user feedback and collective user behavior.

collect data

			
	?	?	3
	5	5	?
	4.5	?	4

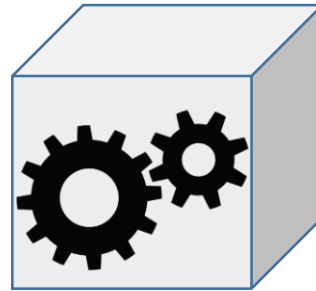
user-movie matrix $R^{M \times N}$

r_{ij} is a rating of i^{th} user for j^{th} movie

? - missing (unknown) values

build model

$$f_U: User \times Item \rightarrow Rating$$



f_U - utility function

generate recommendations



unknown user


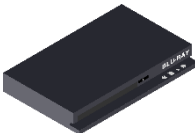

Top-N recommendations Task

The goal is to generate a ranked list of n most relevant items.

$$\text{toprec}(i, n) := \arg \max_j^n \hat{r}_{ij}$$

\hat{r}_{ij} - is the predicted “utility”
of a j -th item to i -th user

Example:

\hat{r}_{ij}	0.73	0.41	0.95
item			

top-2 recommendations



leftsorted list

Remark on User feedback

User feedback is a source for constructing a utility function.

Implicit

- Easy to collect
- Lots of data
- Intrinsic
- Hard to interpret

Explicit

- Hard to collect
- Less data
- Subjective
- “Easy” to interpret

Explicit feedback peculiarities

- horror movies ratings are typically lower, even if user actually likes it



“Ghostbusters” Is A Perfect Example Of How Internet Movie Ratings Are Broken



- IMDb **average user rating**: 4.1 out of 10, of 12,921 reviewers
- IMDb **average user rating among men**: 3.6 out of 10, of 7,547 reviewers
- IMDb **average user rating among women**: 7.7 out of 10, of 1,564 reviewers

Source: <http://fivethirtyeight.com/features/ghostbusters-is-a-perfect-example-of-how-internet-ratings-are-broken/>

Typical problems and challenges

cold-start

- recommendation uncertainty
- representative items

missing values

- 99.99...% of unknowns
- subject to biases, Missing Not at Random (MNAR)

short head / long tail

- 5% of items may hold 40% of all interactions)
- niche products

evaluation

- metric choice
- offline evaluation vs. AB-tests

complex models

- incorporating content information
- including context information

performance

- quick model computation
- real-time recommendations





Filter bubble

<https://www.quora.com/Do-recommender-systems-create-a-“filter-bubble”>

<https://www.quora.com/Should-we-worry-about-the-filter-bubble-created-by-recommender-systems>

<http://open.blogs.nytimes.com/2015/08/11/building-the-next-new-york-times-recommendation-engine/>

Missing values

			
	?	?	3
	5	5	?
	4.5	?	4

? - missing (unknown) values

Typical sparsity level : **99%**
In high dimensional problems: **99.99...%**

However, we don't have to uncover all hidden entries.

Our goal is not matrix completion (even though it can be a proxy)!

Long tail

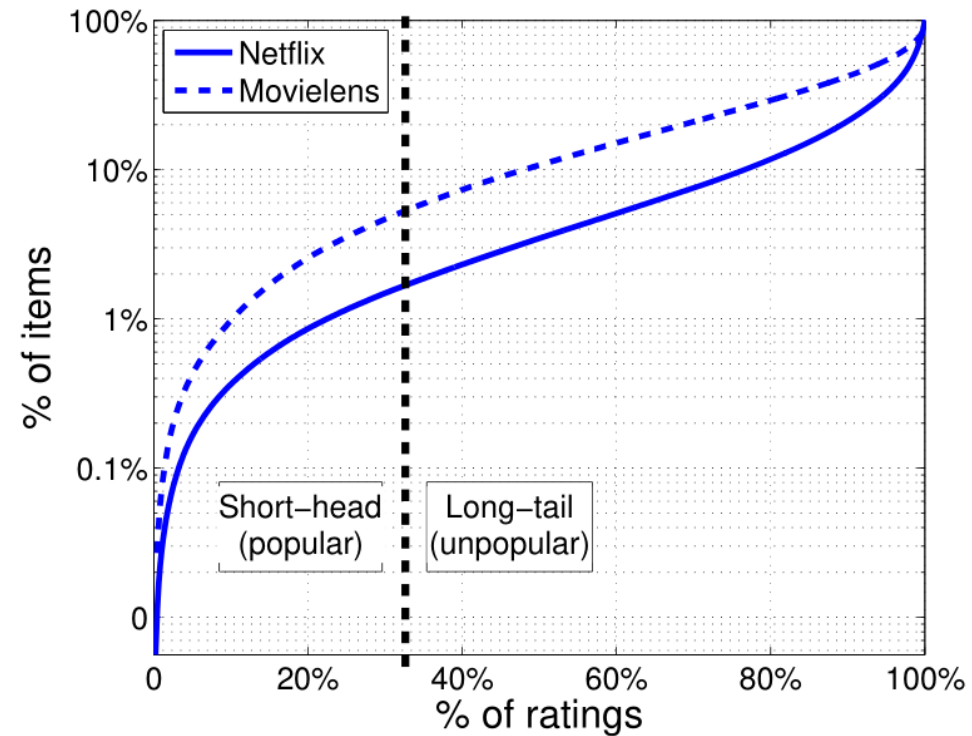
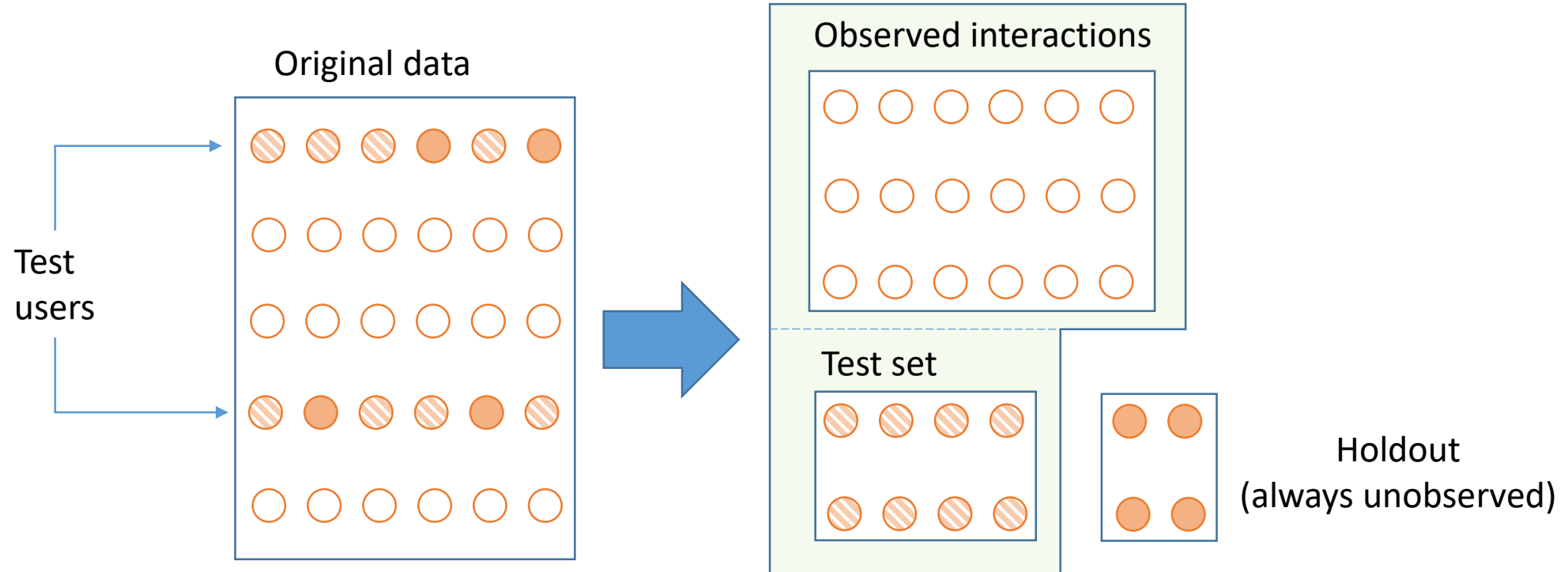


Figure 1: Rating distribution for Netflix (solid line) and Movielens (dashed line). Items are ordered according to popularity (most popular at the bottom).

Evaluation methodology



Data is split **by users**, not by interactions.

In standard evaluation scenario Training set = Observed \cup Testset.

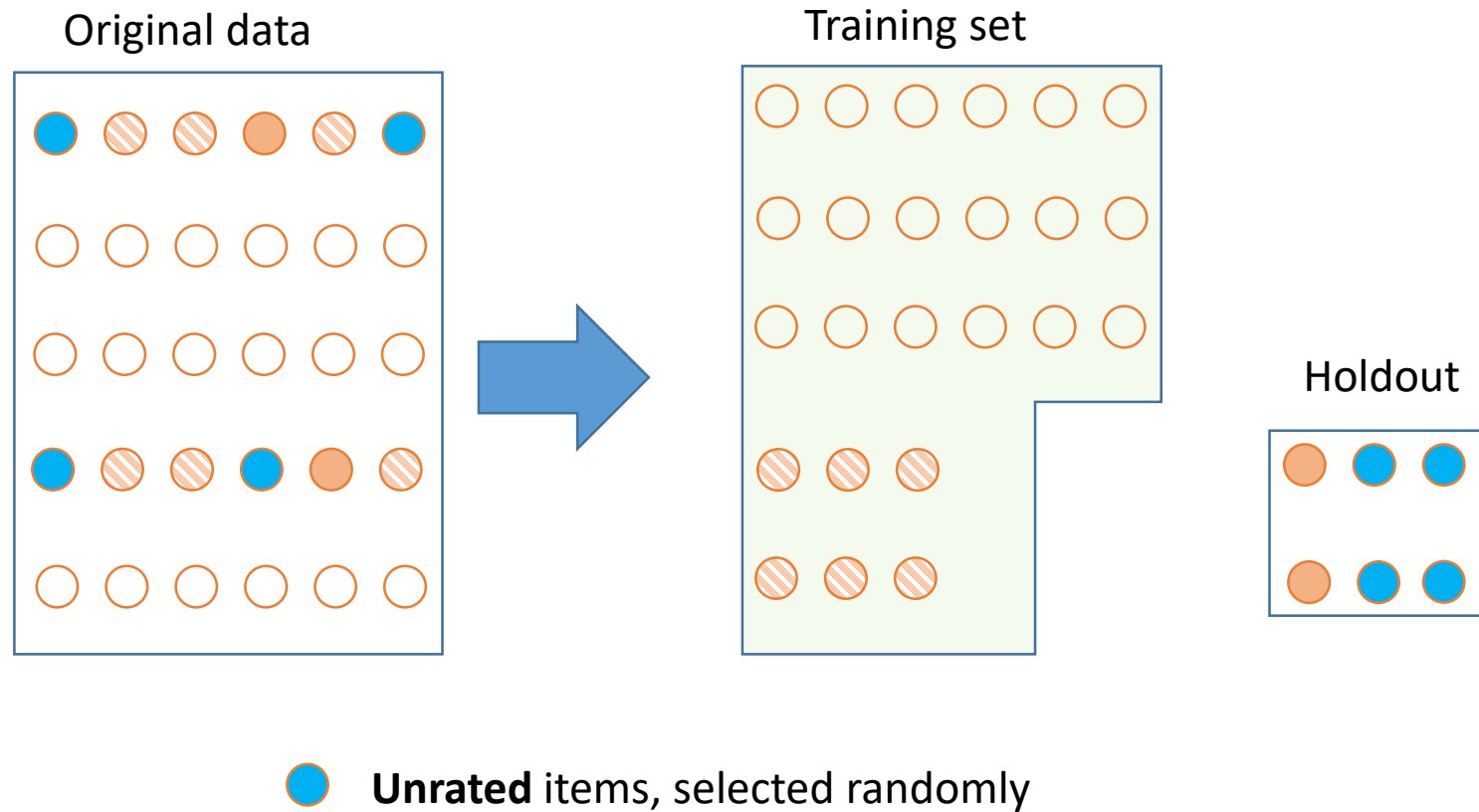
In the “*warm-start*” evaluation scenario the testset is unobserved.

Cross-validation: 80/20.

Holdout can contain top-rated or randomly selected items.

Test and holdout sets share the same users.

Alternative evaluation



Question: what is the difference from the classification problem?

Metrics

Error-based

- RMSE, MAE



Relevance based

- precision, recall
- F1-score
- HR (hit rate)

Ranking-based

- nDGC (normalized discounted cumulative gain),
- MAP (mean average precision),
- MRR (mean reciprocal rank),
- ATOP (area under the TOPK-curve)
- **AUC** (area under the ROC-curve)



Other aspects:

- Coverage
- Novelty
- Serendipity
- Diversity
- Trust
- Utility

Metrics calculation example

$$\text{Precision} = \frac{1}{\#(\text{test users})} \sum_{\text{test users}} \frac{\#(\text{recommended items} \cap \text{holdout items})}{\#(\text{recommended items})}$$

$$\text{Recall} = \frac{1}{\#(\text{test users})} \sum_{\text{test users}} \frac{\#(\text{recommended items} \cap \text{holdout items})}{\#(\text{holdout items})}$$

If $\#(\text{holdout items}) = \text{const} = 1$, then $\text{Recall} \equiv \text{HitRate}$:

$$\text{HR} = \frac{1}{\#(\text{test users})} \sum_{\text{test users}} \text{hit}$$

$$\text{hit} = \begin{cases} 1 & \text{if holdout item is in recommended items,} \\ 0 & \text{otherwise.} \end{cases}$$

$$\text{MRR} = \frac{1}{\#(\text{test users})} \sum_{\text{test users}} \frac{1}{\text{hit rank}}$$

hit rank = position of the item in the recommendations list

Typically computed: $\text{metric}@N$, where $N = \# \text{recommended items}$, e.g. $\text{Recall}@N$, $\text{MRR}@N$, etc.

nDCG calculation

$$DCG = \sum_i \frac{2^{rel_i} - 1}{\log_2(i + 1)} \quad rel_i - \text{true rating of a recommended item at position } i$$

$$nDCG = \frac{DCG}{iDCG}$$

$iDCG = DCG$ with ideal ranking of items

Typically computed: $metric@N$, where $N = \# \text{recommended items}$

Task

Consider top-2 recommender for 10 users from 20-items (2 items per user).

What's better:

Correctly recommend 2 items to each of 5 users

or

1 item to each of 10 users?

Why?

Use standard definition of precision and recall.

Short break

Hands-on time

Simple RS - non-personalized

Idea: Recommend n most popular movies to any user.

This is called “**popularity-based model**”.

There're different ways of calculating popularity of the items.

Look into *polara_intro.ipynb* for implementation details of the most basic approach.

Install Polara framework

<https://github.com/evfro/polara>

Requires: Numpy, Scipy, Pandas, Numba

Install: clone repo and do *python setup.py install*

Use develop branch!

You can also install it directly from source:

```
pip install --upgrade git+https://github.com/Evfro/polara.git@develop#egg=polara
```

Recommendation:

Create separate Anaconda (<https://www.continuum.io/downloads>) environment:

```
conda create -n <your_environment_name> python=3.6 --file conda_req.txt
```

conda_req.txt is a file with all dependencies (including matplotlib, etc.)

See <http://conda.pydata.org/docs/commands/conda-install.html>

Polara

RecommenderData model:

- Data processing, cleaning, preparation
- Rich set of instruments for experimentation pipeline
- Cross-validation controls
- Predictable state
- Can be used for several models simultaneously

RecommenderModel:

- Algorithmic implementations (svd, i2i, etc.)
- Parameters control
- Extensible (build your own model)
- Wrapper for implicit, MyMediaLite, GraphLab
- Controls over RecommenderData instance

Usage examples: <https://github.com/evfro/polara/tree/master/examples>

Your home task

Play with polara framework.

Try to build a better popularity-based model! **You can start with something simple:**

- Top movies by positive ratings only
- Top by mean/median rating
- Weighted sum of positive and negative (ex: $0.8 \cdot pos - 0.1 \cdot neg$)
- Popular but only if drama (or other genre)
- Popular but only if genres count is <less than> or <greater than>...
- Leave some of top movies (e.g. Star Wars)

Or choose your own more creative approach!

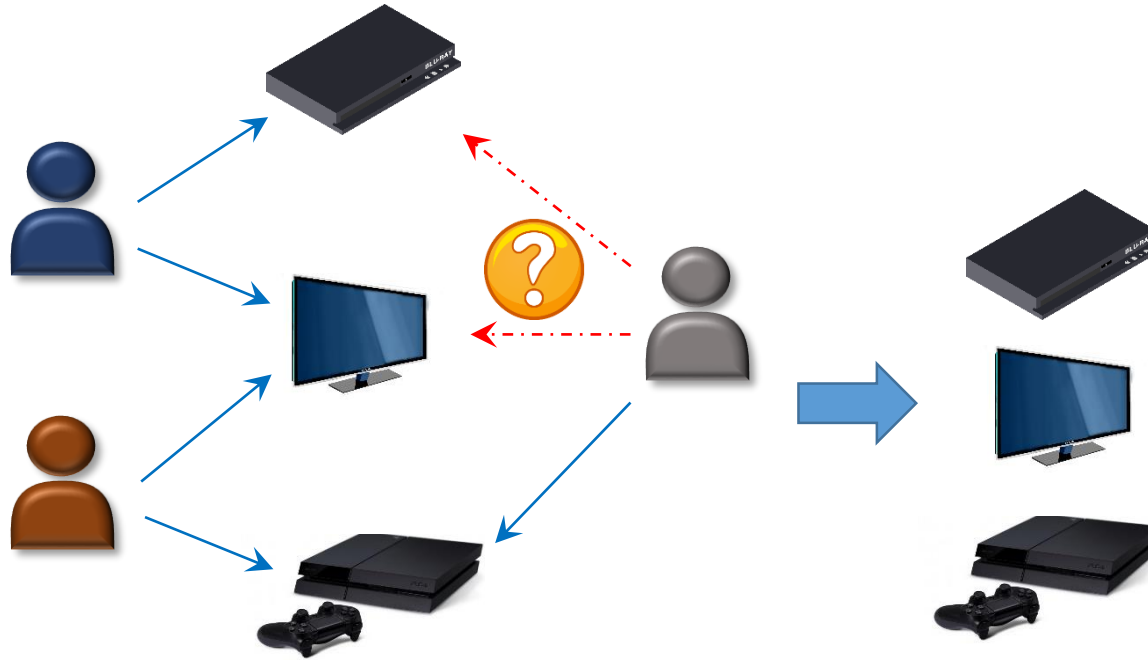
Use standard popularity-based model (by all ratings) as the baseline.

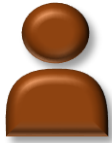


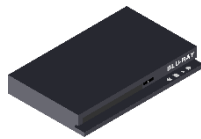









Submit your best model results manually via <http://recsysvalley.azurewebsites.net/upload>
or
programmatically via the provided *polara_intro.ipynb*.

You can view results at:

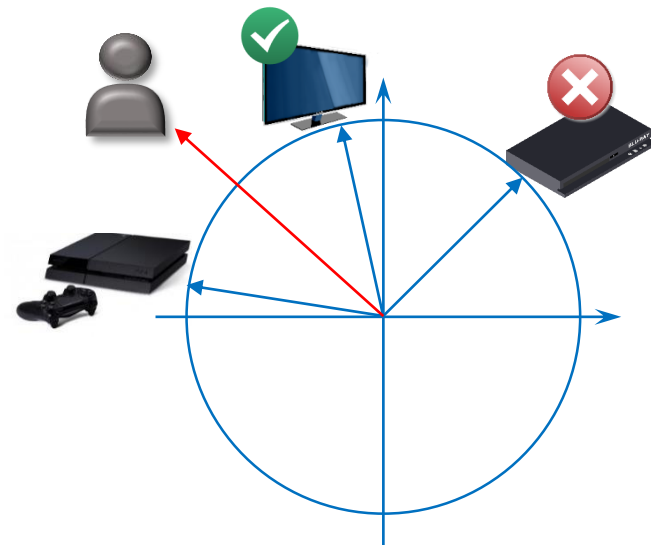
<http://recsysvalley.azurewebsites.net/leaderboard>

Collaborative filtering



$$\text{Sim}(\text{Blu-ray}, \text{PS4}) = \text{Cos}([0, 1, 0], [1, 0, 1]) = 0$$



Non-personalized models

- Popularity-based model (already tried)
- Baseline predictors

Baseline predictors (Biases)

tend to capture much of the observed signal

General bias term:

$$b_{ij} = t_i + f_j + \mu$$

t_i – *tendency* of user i to assign higher or lower rating
(how critical a user is)

f_j – how *f*avorable item j is

μ – global average

Calculation:

“decoupled” form:

$$\begin{cases} t_i = \frac{1}{|I_i| + \beta_i} \sum_{j \in I_i} (r_{ij} - \mu) \\ f_j = \frac{1}{|U_j| + \beta_j} \sum_{i \in U_j} (r_{ij} - t_i - \mu) \end{cases}$$

I_i - set of items, rated by user i ,

U_j - set of users, who rated items j

β_i, β_j - “damping” factors (can set to constant, e.g 25)

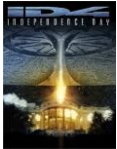
What is the problem here?



5

3

5



3

3

3



Baseline predictors (Biases)

More elaborate way (and more accurate, potentially):

$$\operatorname{argmin} \sum_{i,j \in O} (r_{ij} - t_i - f_j - \mu)^2 + \lambda \left(\sum_i t_i^2 + \sum_j f_j^2 \right)$$

Can be optimized with gradient descent.

O - is a set of all known entries (observed data)

More mathematical way:

$$\operatorname{argmin} \sum_{i,j \in O} (r_{ij} - t_i - f_j)^2 \quad \rightarrow \quad \begin{cases} D\mathbf{t} + S\mathbf{f} = \mathbf{b} \\ D'\mathbf{t} + S^T\mathbf{f} = \mathbf{b}' \end{cases}$$

S - sparsity pattern of the data
 D, D' - diagonal matrices of user-nnz and item-nnz
 \mathbf{b}, \mathbf{b}' - user-sum, item-sum of ratings

Can be effectively optimized with GMRES

$$(D' - S^T D^{-1} S)\mathbf{f} = \mathbf{b}' - S^T D^{-1} \mathbf{b}$$

similarly for \mathbf{t}

additional constraint:

$$\mathbf{f} \perp (1, 1, 1, \dots, 1) \text{ (kernel of the system)}$$

Orthogonality constraint gives natural condition:

$$\sum_j f_j = 0$$

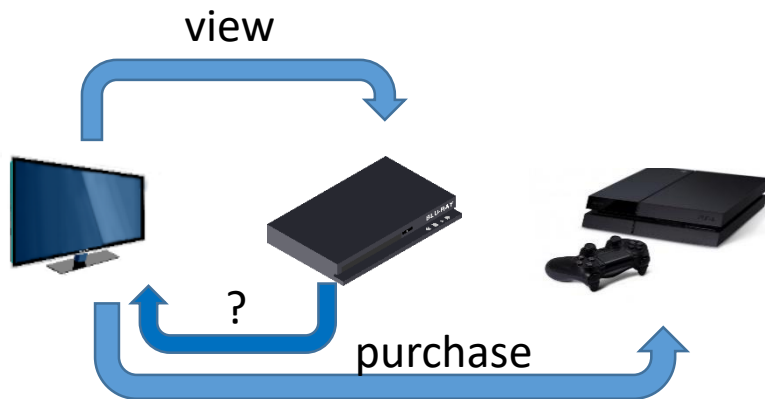
Memory-based techniques

a.k.a neighborhood models or similarity-based models

Pure Item-to-item

- Item-to-item is still a favorite choice in many cases, especially when data is sparse

Count co-occurrence of items:



Pair count:

- Symmetric
- Asymmetric



userid	itemid	transact.
0	575	view
0	1881	view
0	846	basket
1	1878	purchase
1	576	view
...

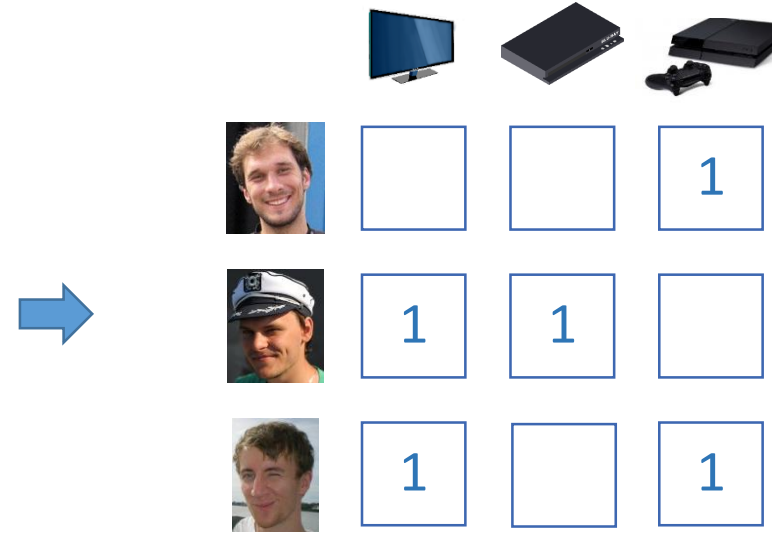
Scaling of values (e.g. count data):

- thresholding
- weighting
- sigmoid
- log*
- TFIDF

Pure Item-to-item

- Convenient representation of the data – sparse matrix

userid	itemid	transact.
0	575	view
0	1881	view
0	846	basket
1	1878	purchase
1	576	view



Can be efficiently stored in CSR or CSC formats.

Also allows for efficient computations
(especially useful for experiments).



How to compute item-to-item co-occurrence matrix
in symmetric case?

How to compute similarity scores in that case?

Computing scores

$$A = R^T R - \text{diag}(R^T R)$$

“self-links”

if p is a vector of known user preferences, then vector of predicted scores is computed as:

$$\text{scores} = Ap$$

Recommendations list:

$$\text{toprec}(i, n) := \arg \max_j^n \hat{r}_{ij}$$

\hat{r}_{ij} - is the predicted score of a j -th item for i -th user

Item-to-item problems

- somewhat obvious recommendations
- low generalization on very sparse data
- no hidden relations modelling

Workaround:

compute cosine-similarity on top of i2i matrix.

But!

matrix becomes dense – increased storage requirements.

Idea:

Maybe limit the number of stored similar items - remember only nearest neighbours?

Remark: i2i matrix can also become dense if there are too many interactions per user.

Amazon item-to-item

2017 [\[PDF\] Amazon.com recommendations item-to-item collaborative filtering](https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf)
<https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf> ▼
Cited by 4095 - Related articles

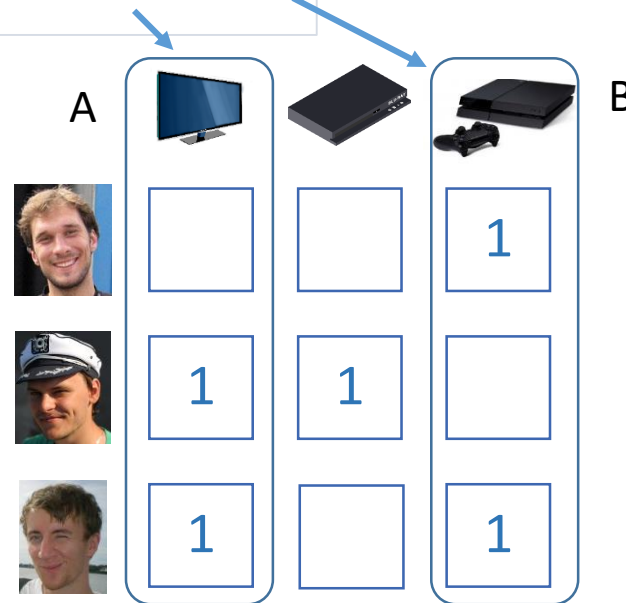
2018 [\[PDF\] Amazon.com recommendations item-to-item collaborative filtering](https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf)
<https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf> ▼
Cited by 5365 - Related articles

```
For each item in product catalog,  $I_1$ 
  For each customer  $C$  who purchased  $I_1$ 
    For each item  $I_2$  purchased by
      customer  $C$ 
      Record that a customer purchased  $I_1$ 
        and  $I_2$ 
  For each item  $I_2$ 
    Compute the similarity between  $I_1$  and  $I_2$ 
```

Iterative algorithm

$$\text{similarity}(\vec{A}, \vec{B}) = \cos(\vec{A}, \vec{B}) = \frac{\vec{A} \bullet \vec{B}}{\|\vec{A}\| * \|\vec{B}\|}$$

Computes similarity of items based on user purchases.



Similarity-based models

a.k.a neighborhood models

Memory-based approach!

Types of models:

- User-based
- Item-based

Scalability trick

$$\text{sim}(i_p, i_q) = \frac{\vec{i}_p \cdot \vec{i}_q}{|\vec{i}_p| \cdot |\vec{i}_q|} = \frac{\sum_{u \in U} r_{u,p} r_{u,q}}{\sqrt{\sum r_{u,p}^2} \sqrt{\sum r_{u,q}^2}}$$

$$\text{sim}(i_p, i_q) = \frac{\text{pairCount}(i_p, i_q)}{\sqrt{\text{itemCount}(i_p)} \sqrt{\text{itemCount}(i_q)}}$$

$$\text{itemCount}(i_p) = \sum r_{u,p} \quad \text{pairCount}(i_p, i_q) = \sum_{u \in U} \text{co-rating}(i_p, i_q)$$

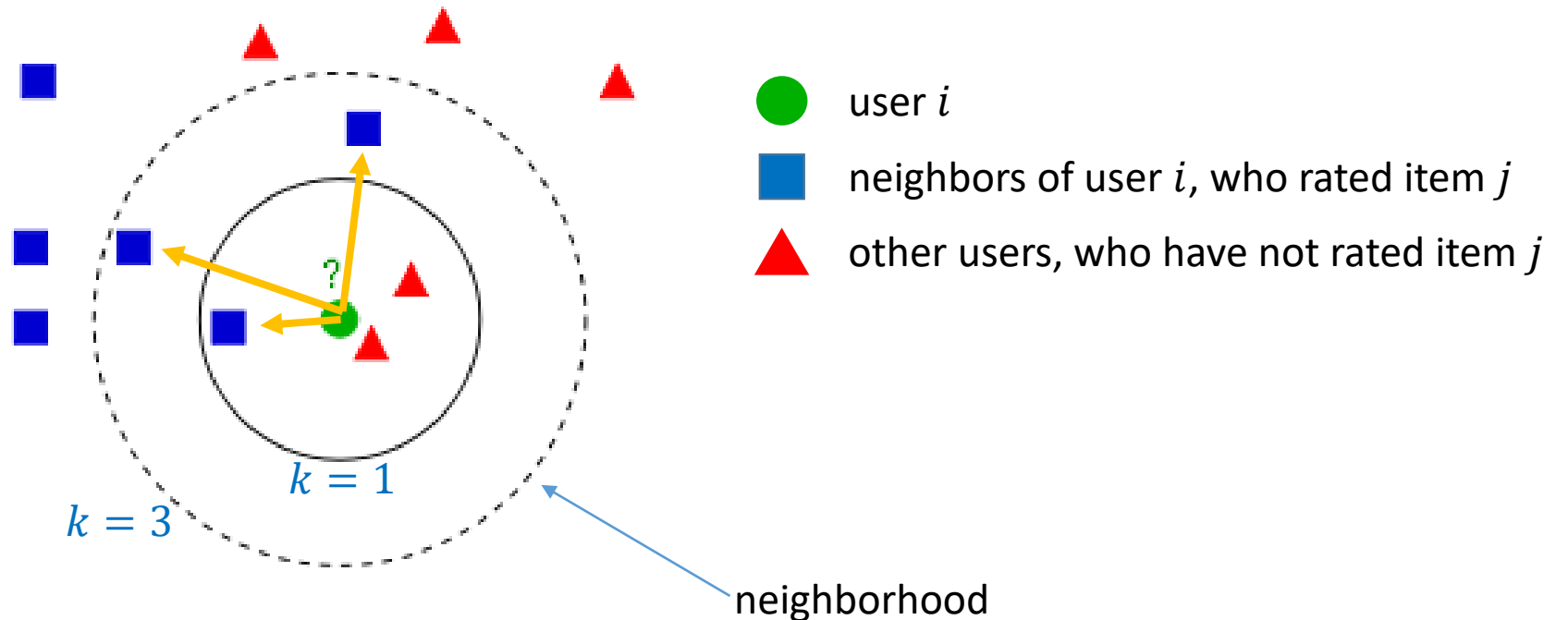
$$\begin{aligned} \text{sim}'(i_p, i_q) &= \frac{\text{pairCount}'(i_p, i_q)}{\sqrt{\text{itemCount}'(i_p)} \sqrt{\text{itemCount}'(i_q)}} = \\ &= \frac{\text{pairCount}(i_p, i_q) + \Delta \text{co-rating}(i_p, i_q)}{\sqrt{\text{itemCount}(i_p) + \Delta r_{u_p}} \sqrt{\text{itemCount}(i_q) + \Delta r_{u_q}}} \end{aligned}$$

User-based models

$$\hat{r}_{ij} = \text{agg } r_{uj} \\ u \in \mathcal{N}_j(i)$$

$\mathcal{N}_j(i)$ denotes *k-nearest-neighbors* (*k*-NN) of user i , who have also rated item j

takes into account opinion
of like-minded users



Neighborhood formation

$$\hat{r}_{ij} = \frac{1}{|U|} \sum_{u \in \mathcal{N}_j(i)} r_{uj}$$

U – set of all users

$$\hat{r}_{ij} = k \sum_{u \in \mathcal{N}_j(i)} \text{sim}(u, i) r_{uj}$$

$\text{sim}(u, i)$ – similarity between users u and i

k – scaling factor $k = \frac{1}{\sum_{u \in \mathcal{N}(i)} |\text{sim}(u, i)|}$ $|\mathcal{N}_i| \approx 20$

Most general form:
$$\hat{r}_{ij} = \bar{r}_i + k \sum_{u \in \mathcal{N}_j(i)} \text{sim}(u, i) (r_{uj} - \bar{r}_u)$$
 \bar{r}_i - average rating of user i

Item-based approach is similar:
$$\hat{r}_{ij} = \bar{r}_j + k \sum_{v \in \mathcal{N}_i(j)} \text{sim}(j, v) (r_{iv} - \bar{r}_v)$$
 $k = \frac{1}{\sum_{v \in \mathcal{N}_i(j)} |\text{sim}(j, v)|}$

Similarity functions: Cosine Similarity, Pearson Correlation, Spearman's Rank Correlation, etc...

When to use user-based or item-based?

- Depends on #users and #items
- Depends on dynamics
- Item-based recommendations are easier to explain.
- User-based recommendations increase serendipity.

Recap: Item-based or user-based approach

Key advantages:

- Easy to implement
- Intuitive explanations
- Good baseline

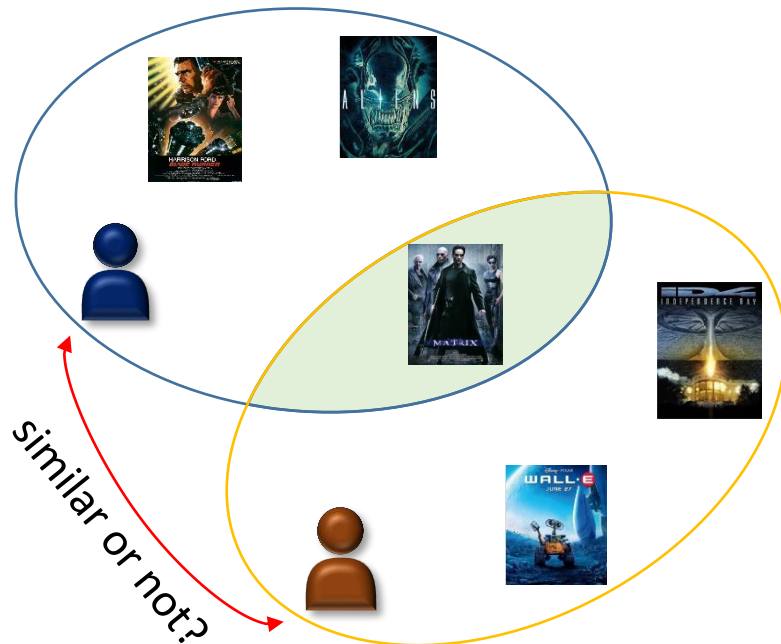
Scalability:

$O(n^2)$ or $O(m^2)$ complexity in the worst case

- due to sparsity real complexity is close to *linear*
- could store only limited number of neighbors
- make incremental updates

Limited coverage problems

Unreliable correlations



Weak generalization

