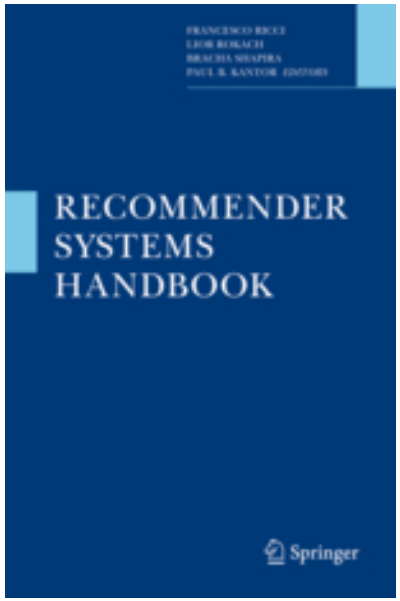
Recommender systems & matrix factorization

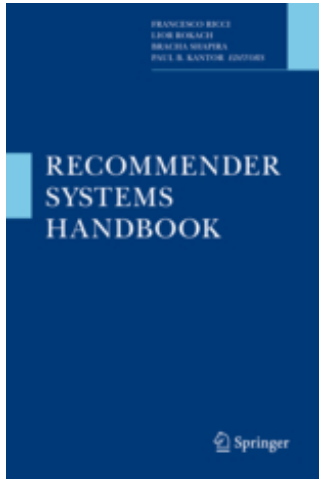
Stéphane Canu
stephane.canu@insa-rouen.fr



APPC, 10 décembre 2023

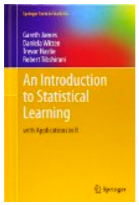
Road map

1. Recommender systems: definitions
2. The Netflix prize
3. Factorize to recommend
4. Evaluation metrics
5. Socially enabled preference learning
   - Tuenti's problem
   - Modeling social preferences
   - Alternating least square minimization
   - Experimental results on real data



Recommendations

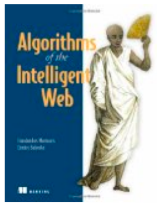
Recommended for You



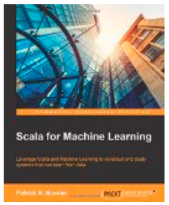
An Introduction to ...
› Daniela Witten
(55)
~~$79.99~~ $73.58
Why recommended?

Interactive Data ...
› Scott Murray
(42)
~~$39.99~~ $26.85
Why recommended?

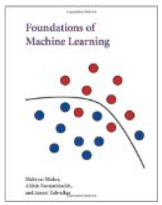
Data Smart: Using ...
› John W. Foreman
(66)
~~$45.00~~ $30.02
Why recommended?



Algorithms of the ...
› H. Marmanis
(16)
~~$44.99~~ $29.13
Why recommended?



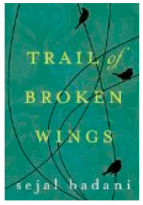
Scala for Machine ...
Patrick R. Nicolas
(6)
~~$59.99~~ $53.99
Why recommended?



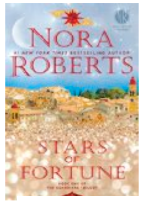
Foundations of Machine .
› Mehryar Mohri
(7)
~~$74.00~~ $66.60
Why recommended?

Hot New Releases in Kindle eBooks

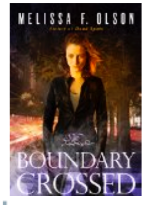
New Release
The Stranger
Harlan Coben

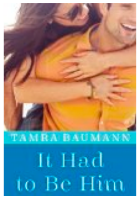


New Release
Trail of Broken Wings
Sejal Badani



S
N
$



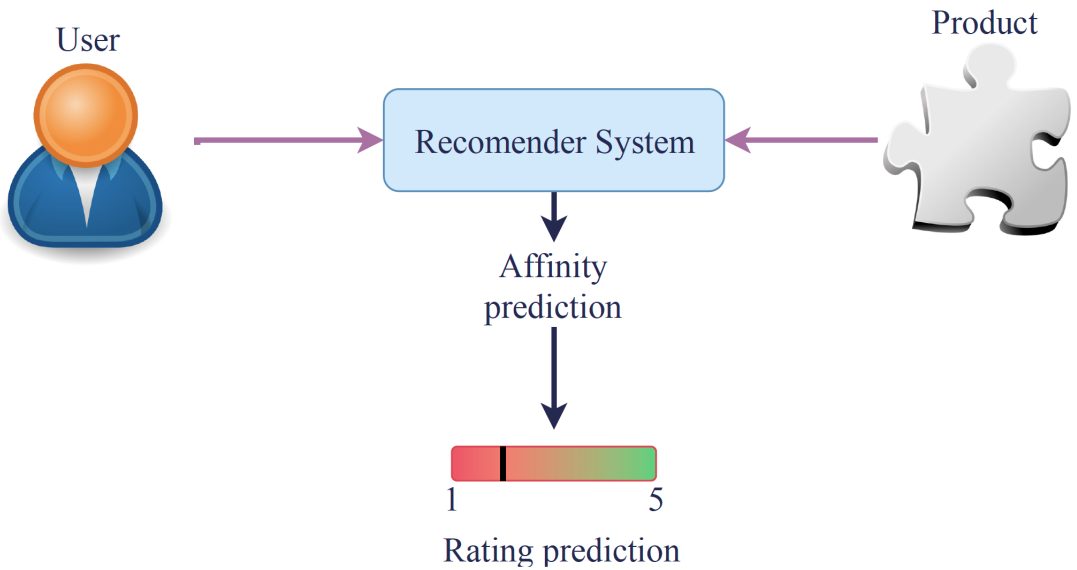


Recommend items, movies, lines of code...

Rec Sys definition

Definition: recommender system

- given a user,
  - given a request of the user on a list of items
- recommend one or more items (suggest relevant items to users)
  - predict the "rating" (or "preference") this user would give to this list

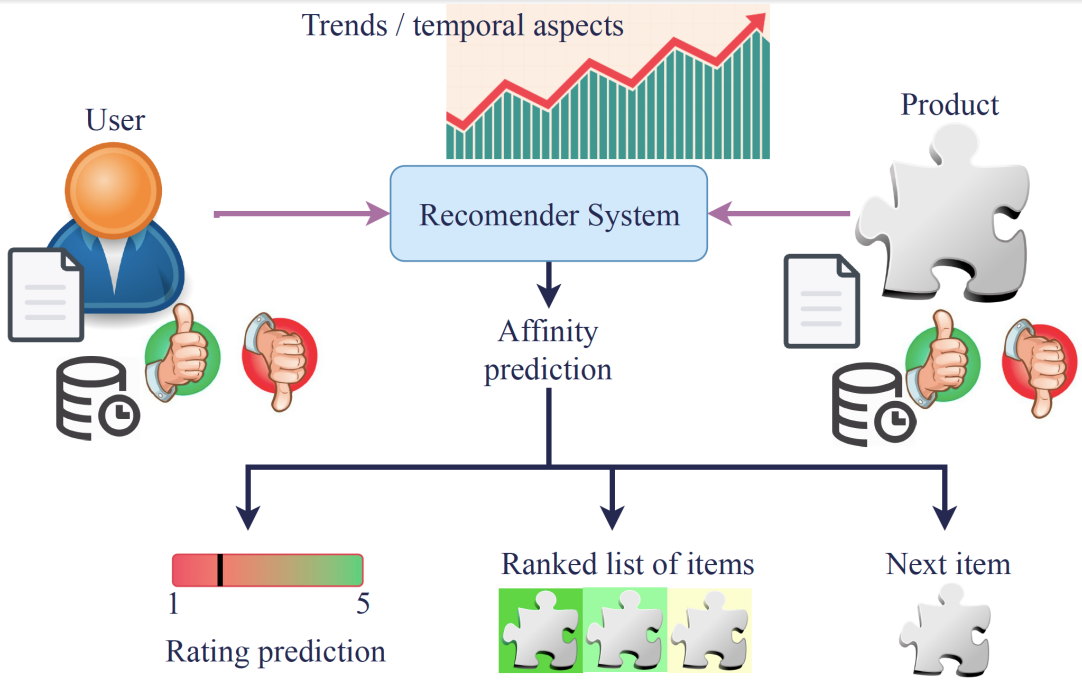


credit V. Guigue, LIP6

Rec Sys definition

Definition: recommender system

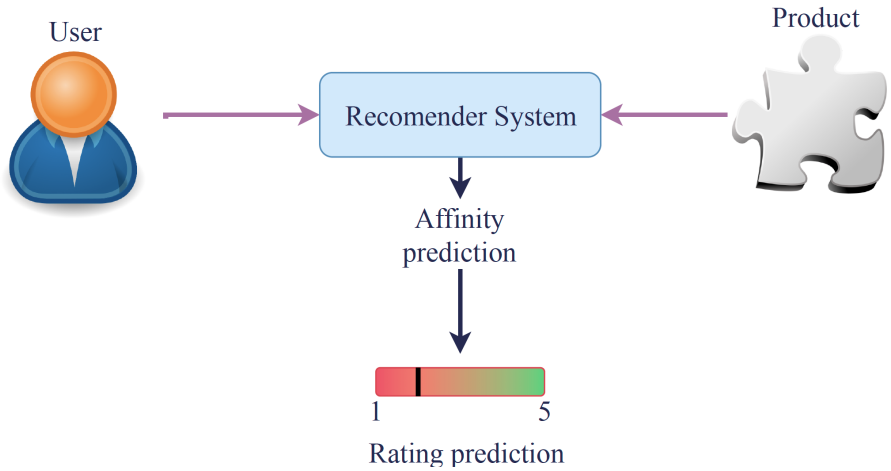
- given a user, data on the user, a list of items, data on these items, some past relations between users and items
  - given a request of the user on a list of items
- recommend one or more items (suggest relevant items to users)
  - predict the "rating" (or "preference") this user would give to this list
  - based on that, select



Data based Recommender system

Available data

- Past behavior
  - what you buy
  - how you like
- Relations to other users
- Item similarity
- Context
  - Time,
  - Sequence,
  - Item description,
  - User categorization: age, socio-professional category,



Rec Sys Examples

Examples:

- the Amazon item-to-item recommendation
- if you buy a remote control, think about the battery
- what is the best smartphone -or car- for me

- Google?

Seller's/User's perspective

Seller

- Increase the number of items sold
- Sell more diverse items
- Increase the user satisfaction
  Increase user fidelity
- Better understand what the user wants

User

- Find Some Good Items [precision issue] quickly and/or in a huge catalog
- Find all good items [recall issue]
  - Layers Information Retrieval tasks
- Being recommended a sequence / a bundle
- Just browsing
- Help others (forum profiles)

the value of recommendations

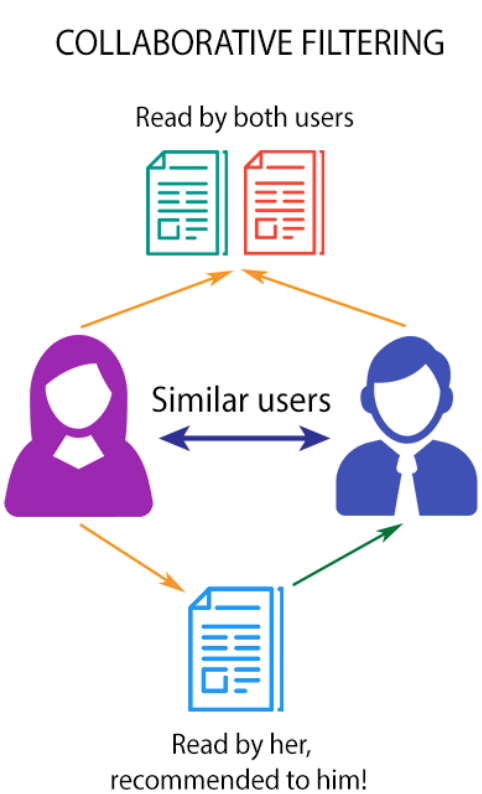
- Amazon: 35% sales from recommendations
- Netflix: 2/3 of the movies watched are recommended
- Google News: recommendations generate 38% more clickthrough
- Choicestream: 28% of the people would buy more music if they found what they liked.

Rec Sys history

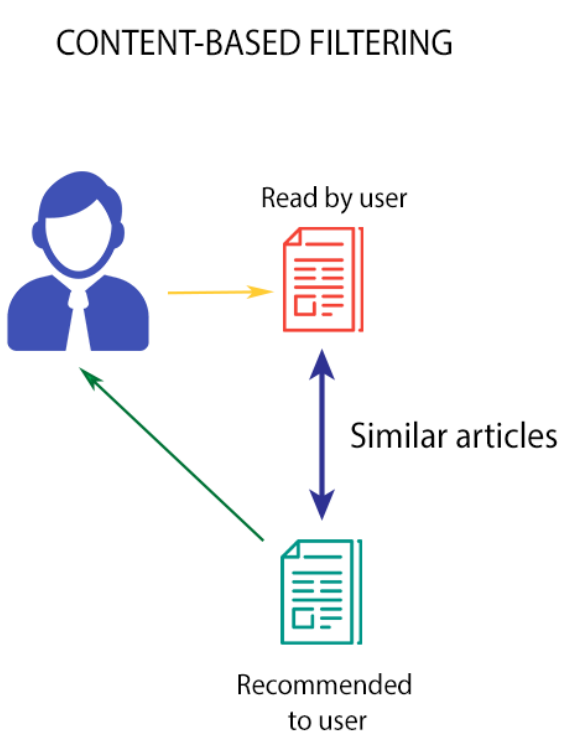
- 1998 Amazon item-to-item recommendation
- 2004-Now Special sessions in recommender system in several important conferences & journals: AI Communications ; IEEE Intelligent Systems; International Journal of Electronic Commerce; International Journal of Computer Science and Applications; ACM Transactions on Computer-Human Interaction; ACM Transactions on Information Systems
- 2007 First ACM RecSys conference
- 2008 Netflix online services (& innovative HMI)
- 2008-09 Netflix RS prize
- 2010-Now RS become essential : YouTube, Netflix, Tripadvisor, Last.fm, IMDb, etc...
- 2014 Recommenders allow access to the long-tail of choices: Discovery

Rec Sys approaches

Collaborative filtering



Item based approach

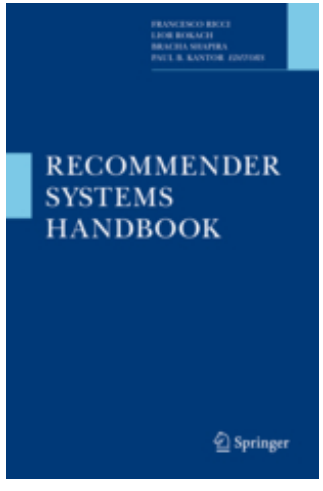


Hybrid Recommendation

combining collaborative filtering, content-based filtering, and other approaches

Road map

1. Recommender systems: definitions
2. The Netflix prize
3. Factorize to recommend
4. Evaluation metrics
5. Socially enabled preference learning
   - Tuenti's problem
   - Modeling social preferences
   - Alternating least square minimization
   - Experimental results on real data



For 1 million $

Netflix Challenge (2007-2009)

- Task: movies recommendation
- CineMatch: Netflix rec. engine
- Improve performances by 10%
- Criterion: square error

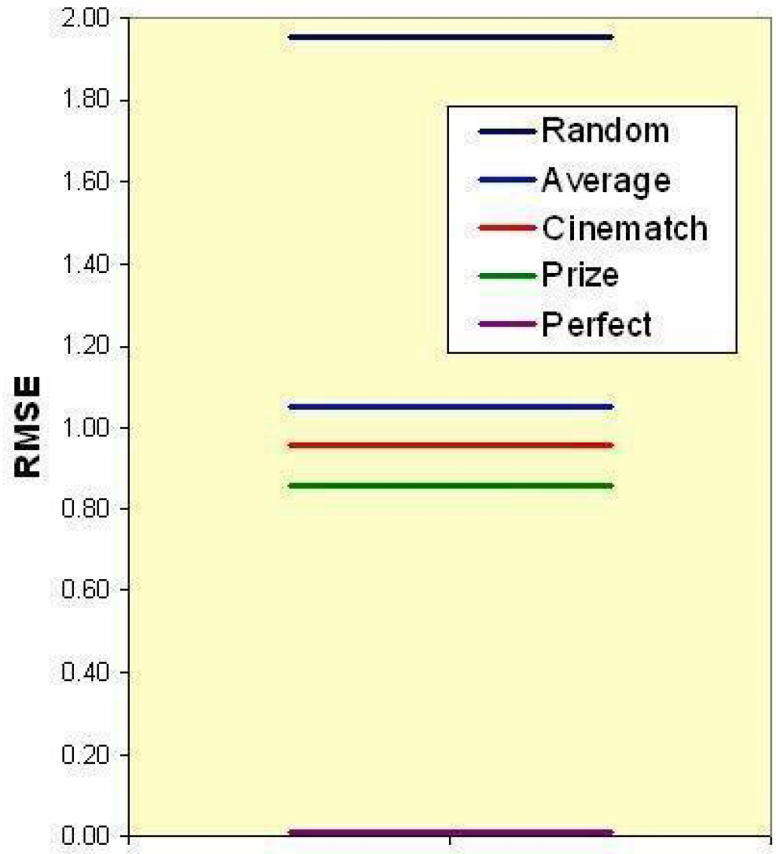
Data

- 480 000 users
- 17 000 movies
- 100 millons scores (from 1 to 5)
- sparcity rate: 1,3 %
- test on (almost) 3 millions (most recent ones)
- 48 000 download

Netflix rules

$$RMSE = \sqrt{\frac{1}{n_t}\sum_{i=1}^{n_t}(y_i - \hat{y}_i)^2}$$

- random: RMSE = 2
- average rating: RMSE = 1.054
- Cinematch: RMSE: 0.9514 (10% improvement)
- prize RMSE: 0.8572 (10% improvement)



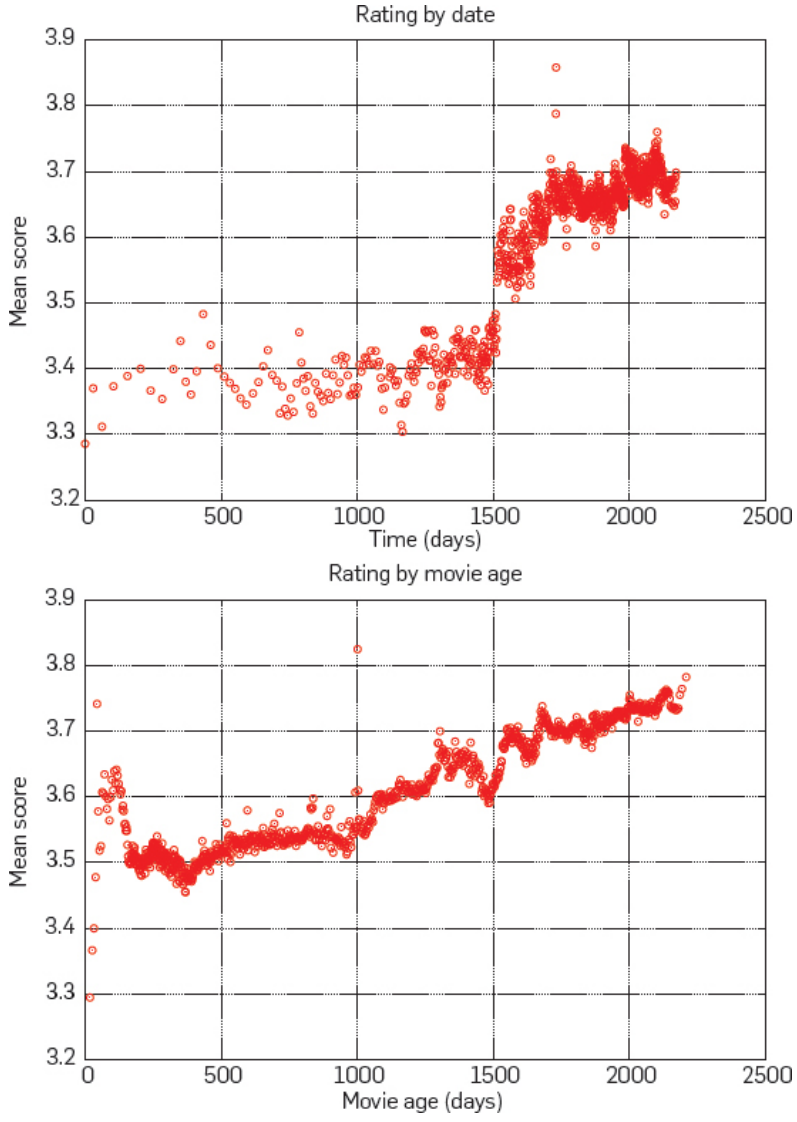
- it is very simple to produce "reasonable" recommendations
- it is extremely difficult to improve them to become "great"

Netfix prize history

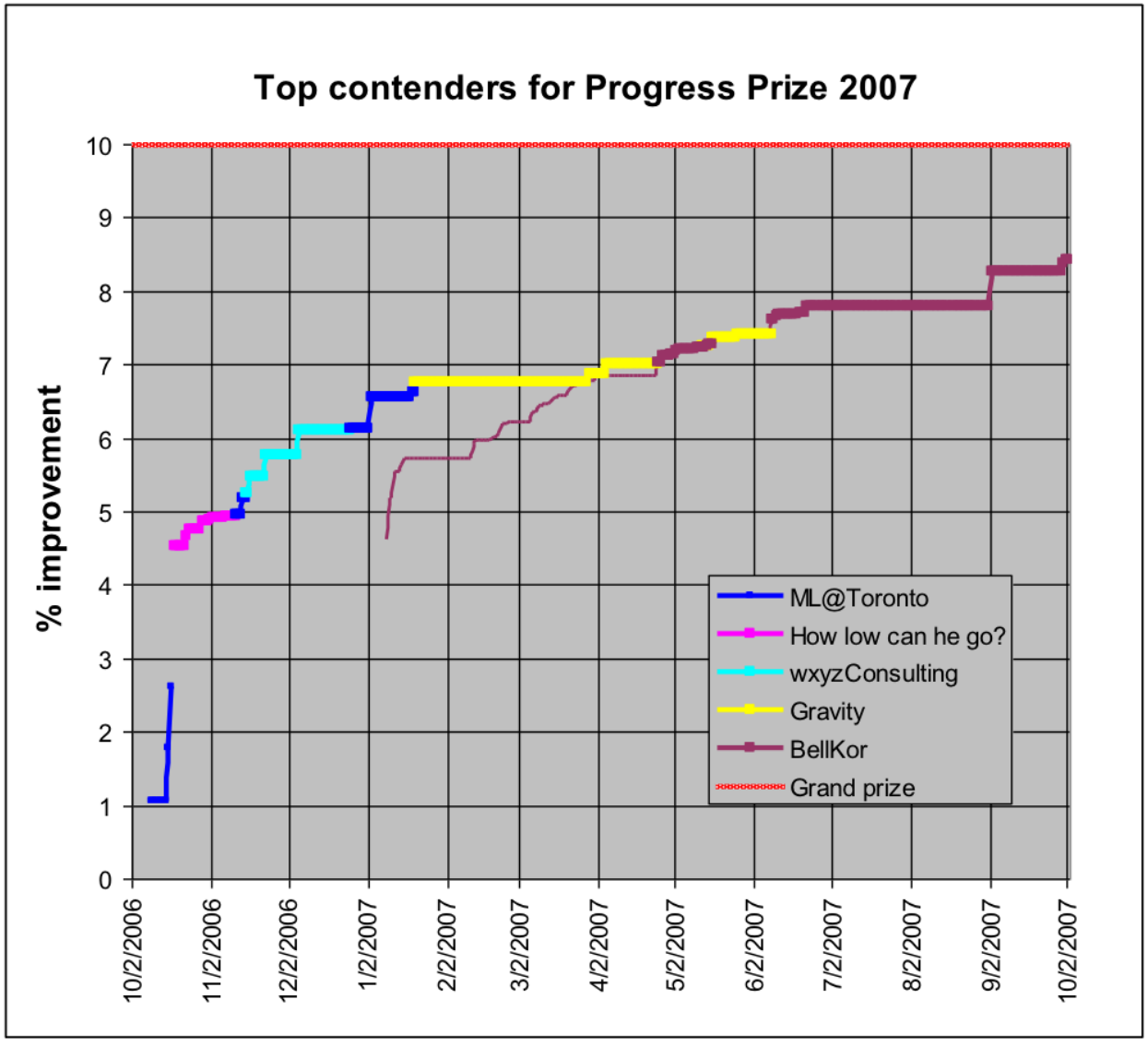
- oct 6, 2006: challenge opened (RMSE: 0.9514)
- oct 2007: team KorBel RMSE: 0.8712 (8.43 % impro.)
  - Top 2 algorithms
    1. Singular Value Decomposition (SVD)
    2. Restricted Boltzmann Machines (RBM as a NN)
- oct 2008: team "BellKor in Big Chaos" RMSE: 0.8616 (9.44 % impr.)
- june 26, 2009: RMSE: 0.8558 (above 10.% impr.)
- july 26, 2009: STOP
  - "The Ensemble" 10.10% improvement on the Qualifying set
  - and "BellKor's Pragmatic Chaos" 10.09% improvement
- sept 18, 2009: the winner is "BellKor's Pragmatic Chaos": prize RMSE: 0.8567 (10.06% improvement) on the hidden test set

Netfix prize data

Averages (global, user, movie) and time effects



Netfix prize leaderboard timeline



Simon Funk's SVD using SGD

- interesting findings during the Netflix Prize came out of a blog post
- incremental, iterative, and approximate way to compute the SVD using gradient descent

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.

$$\min_{U \in \mathbb{R}^{n \times k}, V \in \mathbb{R}^{p \times k}} \sum_{i=1}^{n} \sum_{j=1}^{p} w_{ij} (Y_{ij} - u_i v_j^\top)^2$$

- baseline: CineMatch
- factorization: SVD with d factors $U_i, V_j \in \mathbb{R}^d$

$$\min_{U,V} \sum_{i,j} (Y_{ij} - U_i^\top V_j)^2 + \lambda(\|U_i\|^2 + \|V_i\|^2)$$

  - **normalization** global mean μ, for user i b_i

$$\min_{U,V} \sum_{i,j} (Y_{ij} - U_i^\top V_j - \mu - b_i - p_j)^2 + \lambda(\|U_i\|^2 + \|V_j\|^2)$$

  - **weight** *implicit feedback* c_{ij} **confidence** level

$$\min_{U,V} \sum_{i,j} c_{ij}(Y_{ij} - U_i^\top V_j - \mu - b_i - p_j)^2 + \lambda(\|U_i\|^2 + \|V_j\|^2)$$

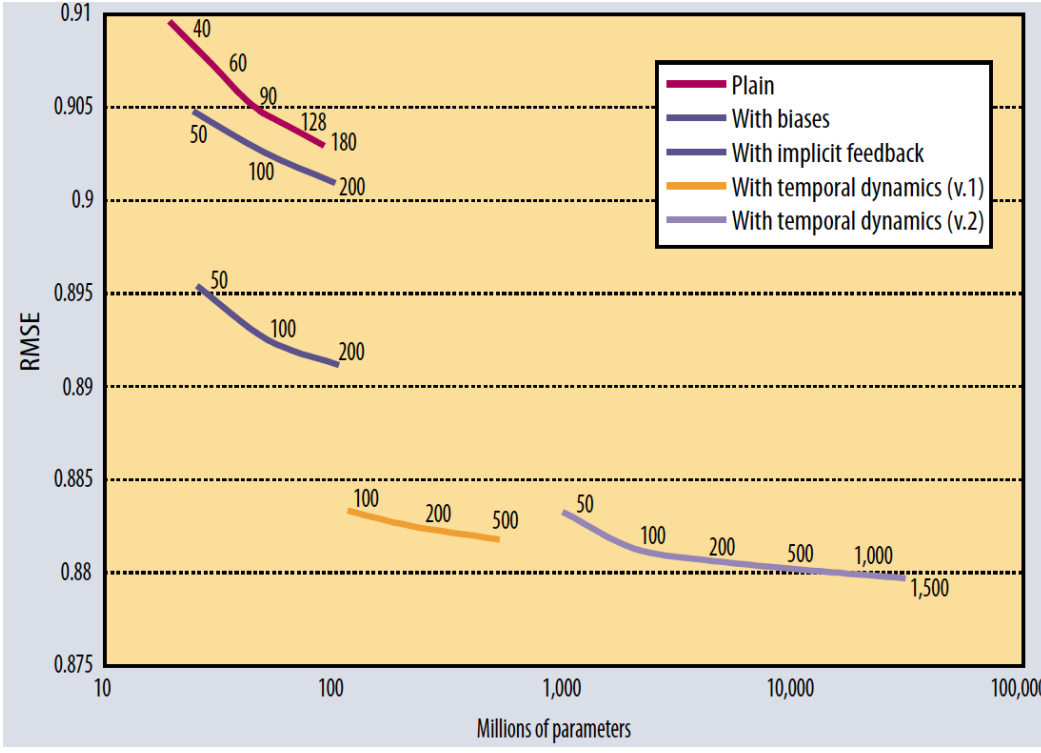
  - **time**

$$\min_{U,V} \sum_{i,j} (Y_{ij} - U_i(t)^\top V_j - \mu - b_i(t) - p_j(t))^2 + \lambda(\|U_i\|^2 + \|V_j\|^2)$$

- mixture of models

Netflix results

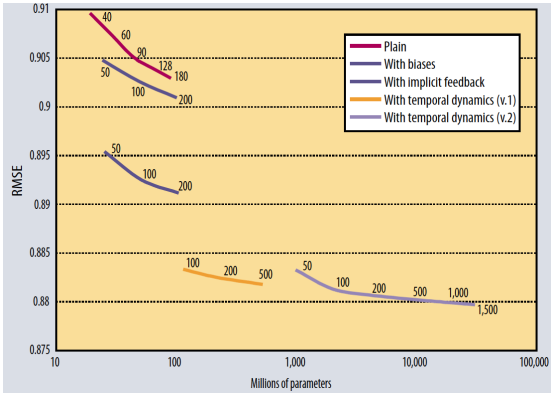
- initial error: 0.9514
- factorization - 4 %
- number of factors - .5 %
- improvements - 2.5 %
  - normalization
  - implicit feedback
  - time
- bagging - 3 % = 0.8563
  mixing 100 methods



Koren, Y., Bell, R.M., Volinsky, C.: Matrix factorization techniques for recommender systems. IEEE Computer 42(8), 30–37 (2009)

Lessons from Netflix

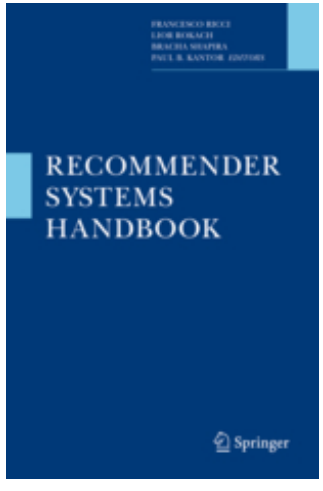
- too specific efforts to be generalized
- factorization *is the solution*
- implicit feedback *helps*
- take into account the specificities (time effect)
- stochastic gradient (that scale)
  - flexible
  - scalable



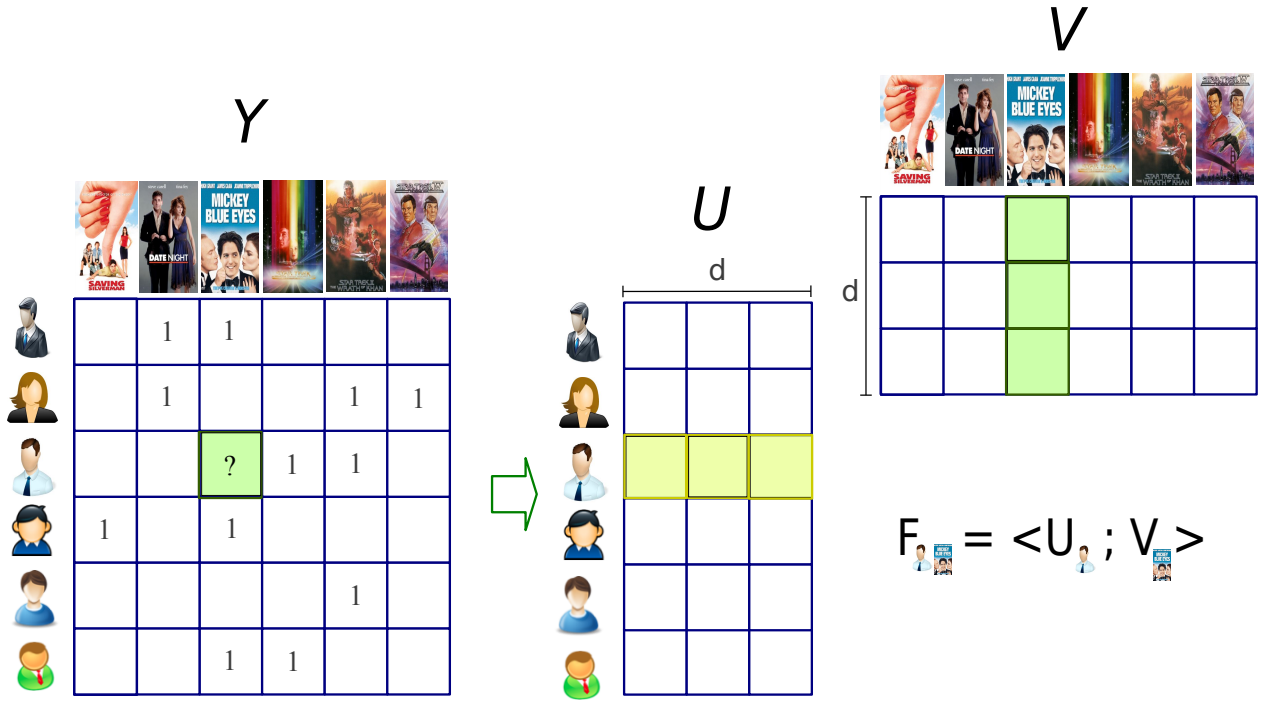
- Currently in use as part of Netflix' rating prediction, but:
  - Designed for 100M ratings, not for xxxB ratings
  - Not adaptable as users add ratings
  - Performance issues

Road map

1. Recommender systems: definitions
2. The Netflix prize
3. Factorize to recommend
4. Evaluation metrics
5. Socially enabled preference learning
   - Tuenti's problem
   - Modeling social preferences
   - Alternating least square minimization
   - Experimental results on real data



Recommendation as matrix completion



$\langle U_i, V_j \rangle$ estimates the *desire* of user i for the product j

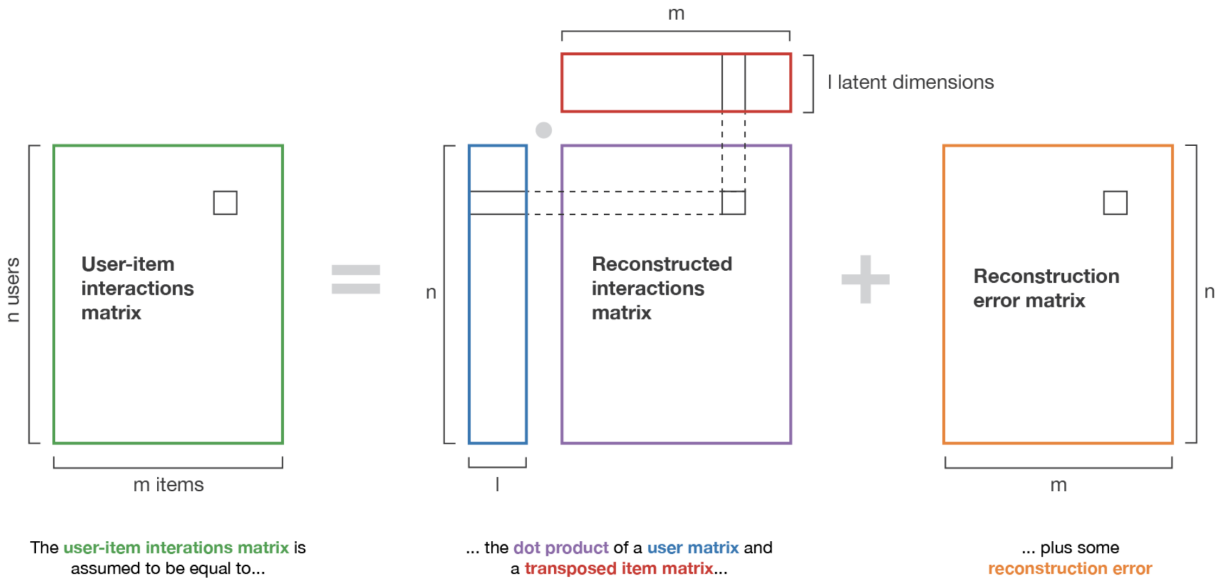
- $U(:,1)$ and $V(1,:)$ are about western
- $U(:,2)$ and $V(2,:)$ are about romance
- ...
- $U(:,d)$ and $V(d,:)$ are about SF

Problem

Model

$$\begin{array}{ccccc} Y & = & F & + & R \\ observation & = & information & + & noise \end{array}$$

F regular and R noise **matrix**



Problem

Model

$$\begin{array}{ccccc} Y & = & F & + & R \\ observation & = & information & + & noise \end{array}$$

F regular and R noise **matrix**

Problem: build $\widehat{F}$ to estimate F

$$\min_{\widehat{F}} \underbrace{\mathcal{L}(Y, \widehat{F})}_{\text{data loss}} + \lambda \underbrace{\Omega(\widehat{F})}_{\text{penalization}}$$

- for λ well chosen
- $\widehat{F} = UV^t$ low rank factorization

How to estimate U et V?

Low rank matrix Y approximation

$$\begin{aligned} \min_{\widehat{F}} \quad & \mathcal{L}(Y, \widehat{F}) := \|Y - \widehat{F}\|_F^2 = \sum_{i=1}^{n} \sum_{j=1}^{p} (Y_{ij} - F_{ij})^2 \\ \text{with} \quad & \text{rank}(\widehat{F}) = d \end{aligned}$$

Eckart & Young 1936

Solution: principal component analysis (PCA) $\widehat{F} = UV^t$

$$\begin{aligned} & Y_n = normalise(Y) \\ & C = Y_n' * Y_n \\ & d = 3 \\ & V, \Sigma = \text{eig}(C, d) \\ & \widehat{F} = UV' \end{aligned}$$

How to estimate U et V?

Low rank matrix Y approximation

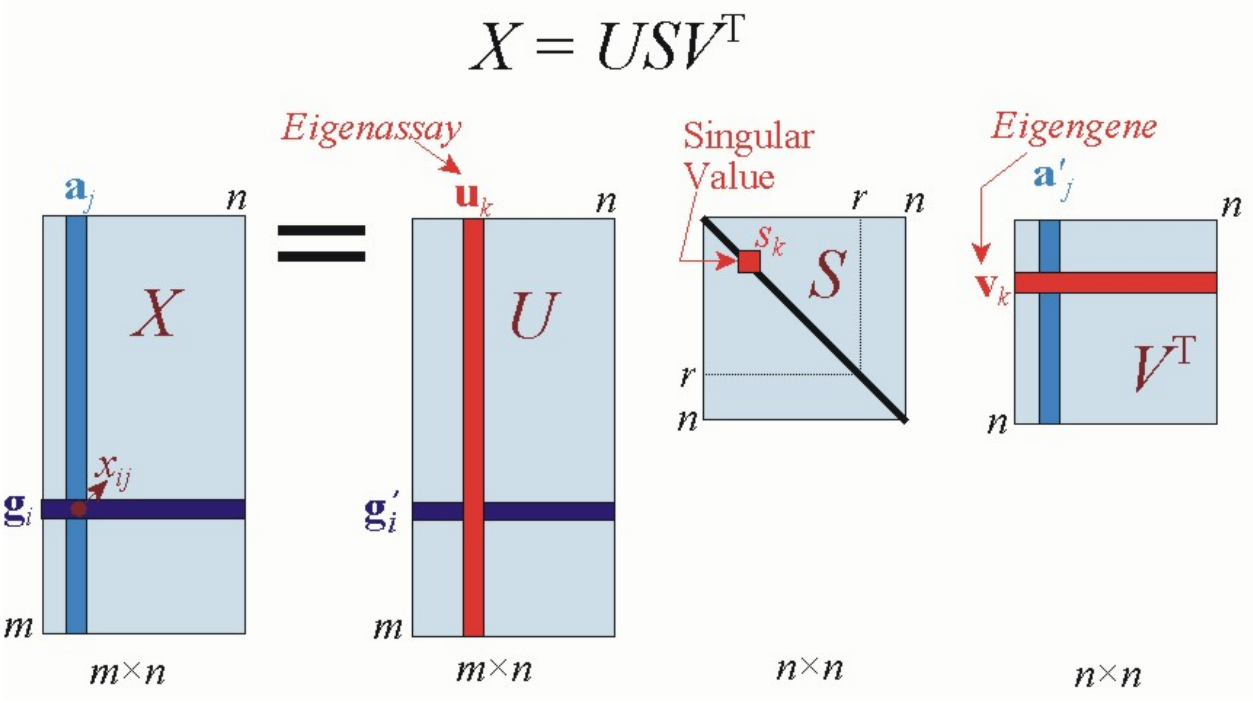
$$\begin{aligned} \min_{\widehat{F}} \quad & \mathcal{L}(Y, \widehat{F}) := \|Y - \widehat{F}\|_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{p} (Y_{ij} - F_{ij})^2 \\ \text{with} \quad & \text{rank}(\widehat{F}) = d \end{aligned}$$

Eckart & Young 1936

Solution: principal component analysis (PCA) $\widehat{F} = UV^t$

$Y_n = normalise(Y)$	Large scale adaptation
$C = Y_n' * Y_n$	OUT OF MEMORY
$d = 3$	we need more 200
$V, \Sigma = \text{eig}(C, d)$	full $\rightarrow$ sparse SVD
$\widehat{F} = UV'$	

PCA, eigenvalues and SVD



$$X^\top X = VS^2V^\top$$

$$XX^\top X = US^2U^\top$$

How to implement PCA (to retrieve U and V)?

Singular value decomposition (SVD) $\quad \widehat{F} = U^t V \quad (= \widetilde{U} \Sigma V^t)$

Exact solution on sparse matrices ($??? = 0$)

- svd too slow
- use adapted Lanczos process
  - bidiagonalization $Y = WBZ$
  - diagonalize B (using Givens rotations GVL 8.6.1)
- adapted software: svds, PROPACK [a]

[a] http://soi.stanford.edu/ rmunk/PROPACK/

```
>> size(Y)          ans =       480189        17770
>> d = 200;
>> tic
>> [U,D,V] = lansvd(Y,d,'L');
>> toc                Elapsed time is 1030.8 seconds.
```

217 sec at https://dgleich.wordpress.com/2013/10/19/svd-on-the-netflix-matrix/

How to compute U and V ?

$$\min_{U,V} \mathcal{L}(Y, UV) \qquad \text{with} \qquad rang(U) = rang(V) = k$$

SGD (Stochastic Gradient Descent)

- Compute the gradients
- loop
  - $U_i^{NEW} = U_i^{OLD} - p\nabla_{U_i}\mathcal{L}(Y, U, V)$
  - $V_j^{NEW} = V_j^{OLD} - p\nabla_{V_j}\mathcal{L}(Y, U, V)$

Existing implementations

- S. Funk process[a], PLSA, MMMF (Srebro et al, 2004), ...
- code : RSVD[b], CofiRank[c], ...

[a] http://sifter.org/ simon/journal/20061211.html
[b] http://code.google.com/p/pyrsvd/
[c] https://github.com/markusweimer/cofirank

Weighted Singular Value Decomposition algorithm

$$\min_{U \in \mathbb{R}^{n \times k}, V \in \mathbb{R}^{p \times k}} J_w(U,V) \quad \text{with} \quad J_w(U,V) = \sum_{i=1}^{n} \sum_{j=1}^{p} w_{ij} (Y_{ij} - u_i v_j^\top)^2$$

where $w_{ij} = 0$ if Y_{ij} is unknown (and else $w_{ij} = 1$)

Marlin's (or Srebro and Jaakkola) Expectation Maximization (EM) algorithm. Missing values are completed with values that does not change the results

$$Z = Y + ???$$

$$\begin{aligned} Z &= WY + (1 - W)Z \\ U, V &= svd(Z) \\ Z &= UV^\top \end{aligned}$$

Z in no longer sparse

Alternated optimization

General framework

- $\min_{U,V} \mathcal{L}(U, V, Y) + \Omega(U, V)$
  - data related loss: $\mathcal{L}(U, V, Y)$
  - penalization term: $\Omega(U, V)$

scalable algorithm

- compute partial gradients
- example:

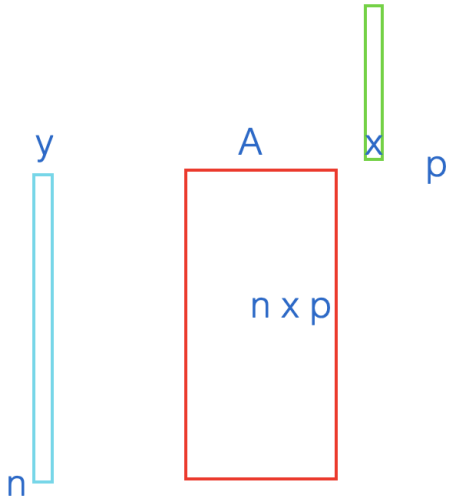
  for $L(U, V, Y) = \|UV - Y\|^2$ and $\Omega(U, V) = 0$

  we have: $\rightarrow U_i = (Y_i V^\top).(VV^\top)^{-1}$

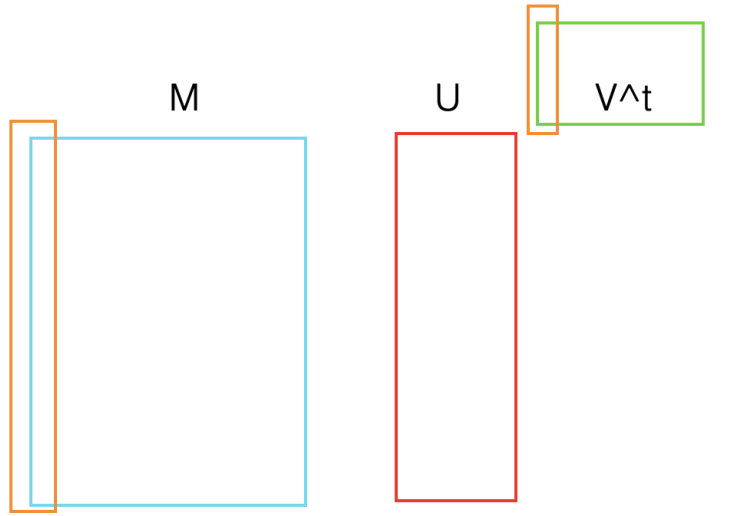
Alternated optimization

min_U ||M^t - V U^t||_W^2 (+ lam ||U||^2)

min_x ||y - Ax||_W^2 = \sum_i w_i (y_i - A_i x)^2



min_V ||M - U V^t||_W^2 ~~(+ lam ||V||^2)~~



Soit W = diag(w) (une matrice de taille n x n)

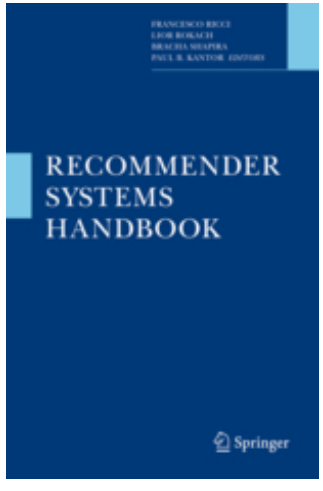
la solution du problème s'écrit : x_s = (At W A)^{-1} At W y

soit en python x = linalg.sovle(A.t@W@A,A.t@W@y)

si on ajoute + lam ||x||^2
la solution devient x_r = (At W A + lam I)^{-1} At W y
ou I est la matrice identité de taille p x p

Road map

1. Recommender systems: definitions
2. The Netflix prize
3. Factorize to recommend
4. Evaluation metrics
5. Socially enabled preference learning
   - Tuenti's problem
   - Modeling social preferences
   - Alternating least square minimization
   - Experimental results on real data



How good is the RMSE?

- RMSE (Root Mean Squared Error) is typically used to evaluate regression problems
- It tends to disproportionately penalize large errors as the residual (error term) is squared. This means RMSE is more prone to being affected by outliers or bad predictions.
- Fails at taking the Long Tail into account

Mean Average Precision (from the IR domain)

Precision at k ($p@k$)is the proportion of recommended items in the top-k set that are relevant

For a user u with 4 liked items to discover:

$$prediction = \begin{matrix} i_{12} \\ i_8 \\ i_{42} \\ i_1 \end{matrix} \quad \begin{matrix} i_1 \\ i_{42} \\ i_8 \\ i_9 \end{matrix} = GT$$

$$p@4 = \frac{3}{4}$$

Average Precision

$$\frac{1}{K}\sum_{k=1}^{K} p@k = \frac{1}{4}(p@1 + p@2 + p@3 + p@4) = \frac{1}{4}(0 + 0 + \frac{2}{3} + \frac{3}{4})$$

Mean Average Precision: mean over the population

Mean Average Recall (from the IR domain)

Recall at k ($r@k$)is the number of recommended items in the top-k set that are relevant divided by the number of relevant items
For a user u with 4 liked items to discover:

$$prediction = \begin{matrix} i_{12} & i_1 \\ i_8 & i_{42} \\ i_{42} & i_8 \\ i_1 & i_9 \end{matrix} = GT$$

$$r@3 = \frac{2}{4}$$

Average Recall

$$\frac{1}{K}\sum_{k=1}^{K} p@k = \frac{1}{4}(r@1 + r@2 + r@3 + r@4) = \frac{1}{4}(0 + 0 + \frac{2}{4} + \frac{3}{4})$$

Mean Average recall: mean over the population

Mean Reciprocal Rank

At which rank is the first relevant item?
For a user u with 4 liked items to discover:

$$prediction = \begin{matrix} i_{12} \\ i_8 \\ i_{42} \\ i_1 \end{matrix} \quad \begin{matrix} i_1 \\ i_{42} \\ i_8 \\ i_9 \end{matrix} = GT$$

$$RR = \frac{1}{rank_i} = \frac{1}{2}$$

Mean Reciprocal Rank = Averaging over the whole population

nDCG : Normalized Discounted Cumulative Gain

Assume a relevance score for each item is available:

$$prediction = \begin{matrix} i_{12} \\ i_8 \\ i_{42} \\ i_1 \end{matrix} \begin{matrix} 0 \\ 2 \\ 3 \\ 3 \end{matrix} = \text{relevance} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \end{matrix} = \text{position}$$

Discounted Cumulative Gain k is:

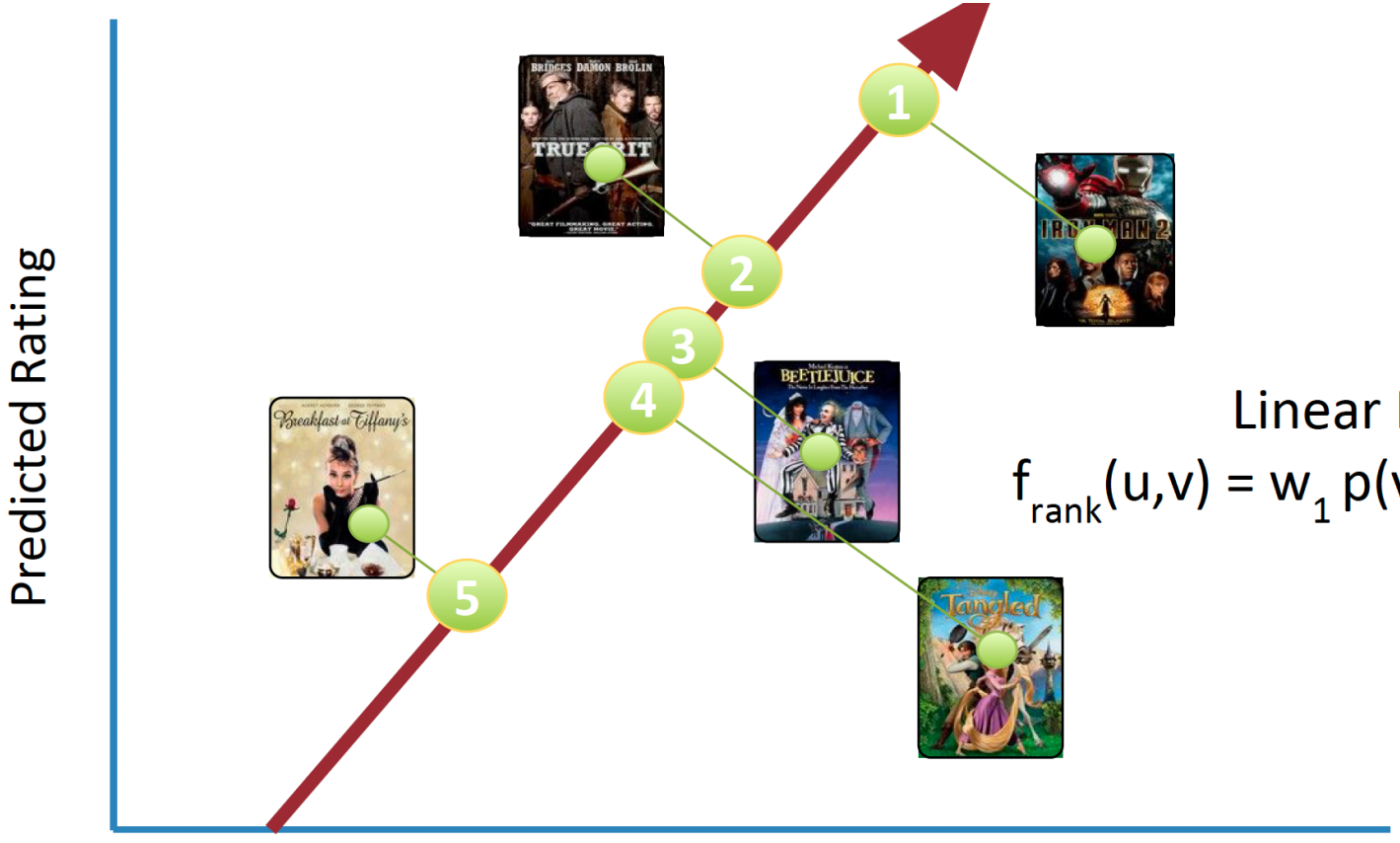
$$DCG@k = \sum_{i=1}^{k} \frac{relev_i}{\log_2(i+1)} = 0 + \frac{2}{log_2(3)} + \frac{3}{log_2(4)} + \frac{3}{log_2(5)} = 4.05$$

$$nDCG@4 = \frac{DCG@4}{IdealDCG@4} = \frac{4.05}{5.89} = 0.69$$

with IdealDCGk relavance being = (3,3,2,0)

NDCG is a measure of ranking quality

Learn to rank



Linear Model:

$$f_{rank}(u,v) = w_1\, p(v) + w_2\, r(u,v) + b$$

e.g CofiRank, RankSVM, ...

Rec Sys in python

surprise

A Python scikit for recommender systems.

Home

Documentation

GitHub page

Star Fork

Maintained by Nicolas Hug
Page built with Jekyll and Hyde

Overview

Surprise is a Python scikit building and analyzing recommender systems that deal with explicit rating data.

Surprise **was designed with the following purposes in mind**:

- Give users perfect control over their experiments. To this end, a strong emphasis is laid on documentation, which we have tried to make as clear and precise as possible by pointing out every detail of the algorithms.
- Alleviate the pain of Dataset handling. Users can use both *built-in* datasets (Movielens, Jester), and their own *custom* datasets.
- Provide various ready-to-use prediction algorithms such as baseline algorithms, neighborhood methods, matrix factorization-based (SVD, PMF, SVD++, NMF), and many others. Also, various similarity measures (cosine, MSD, pearson...) are built-in.
- Make it easy to implement new algorithm ideas.
- Provide tools to evaluate, analyse and compare the algorithms performance. Cross-validation procedures can be run very easily using powerful CV iterators (inspired by scikit-learn excellent tools), as well as exhaustive search over a set of parameters.

The name *SurPRISE* (roughly :)) stands for Simple Python RecommendatIon System Engine.

Please note that surprise does not support implicit ratings or content-based information.

Getting started, example

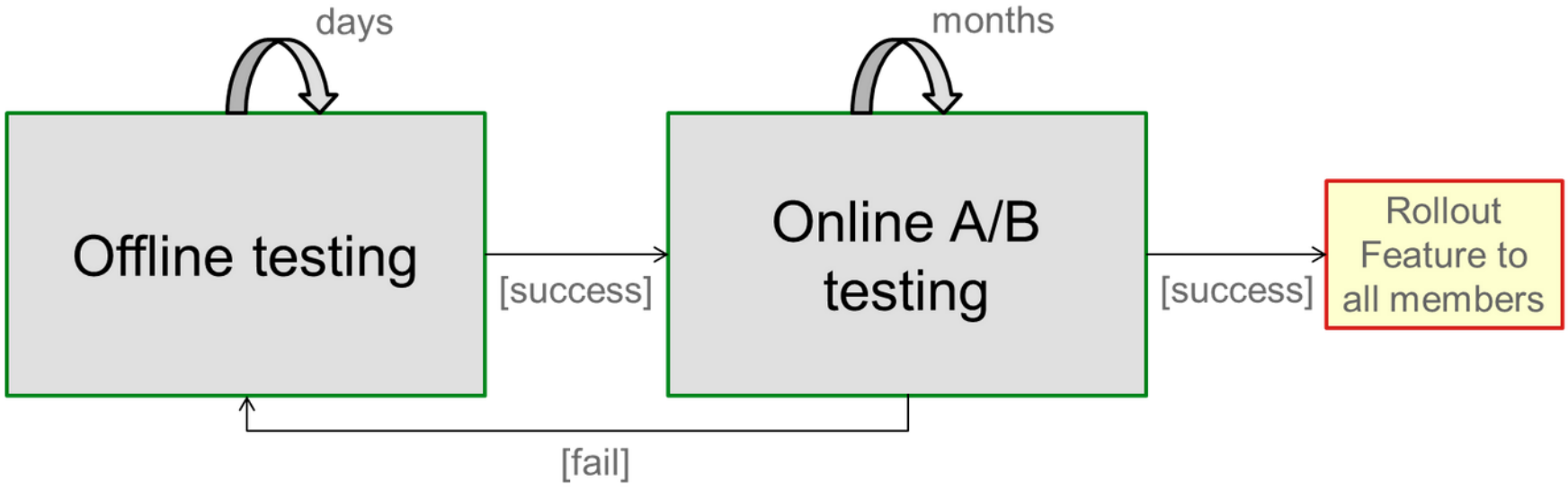
Here is a simple example showing how you can (down)load a dataset, split it for 5-fold cross-validation, and compute the MAE and RMSE of the SVD algorithm.

```
from surprise import SVD
from surprise import Dataset
from surprise.model_selection import cross_validate

# Load the movielens-100k dataset (download it if needed).
data = Dataset.load_builtin('ml-100k')
```

A/B testing & production launch

Divide your online users into A/B-test groups



The easiest measures are the Click-Through Rate (CTR) and the Conversion Rate (CR) of the recommendations.

Road map

1. Recommender systems: definitions
2. The Netflix prize
3. Factorize to recommend
   - Factorization and singular values decomposition
4. Evaluation metrics
5. Socially enabled preference learning
   - Tuenti's problem
   - Modeling social preferences
   - Alternating least square minimization
   - Experimental results on real data

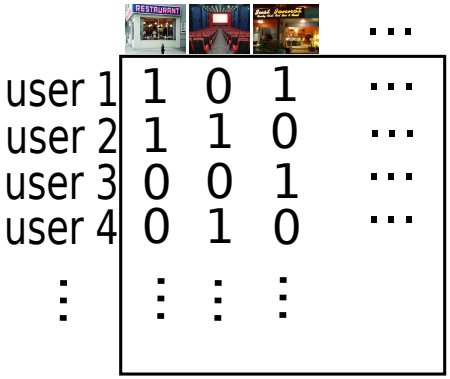
Tuenti's problem

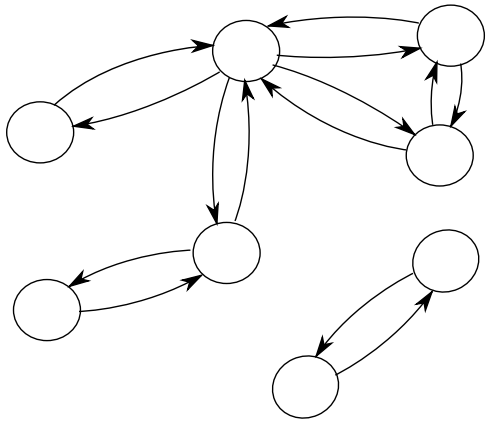
Size

- 13M people
- 100K places
- 200M social connexions

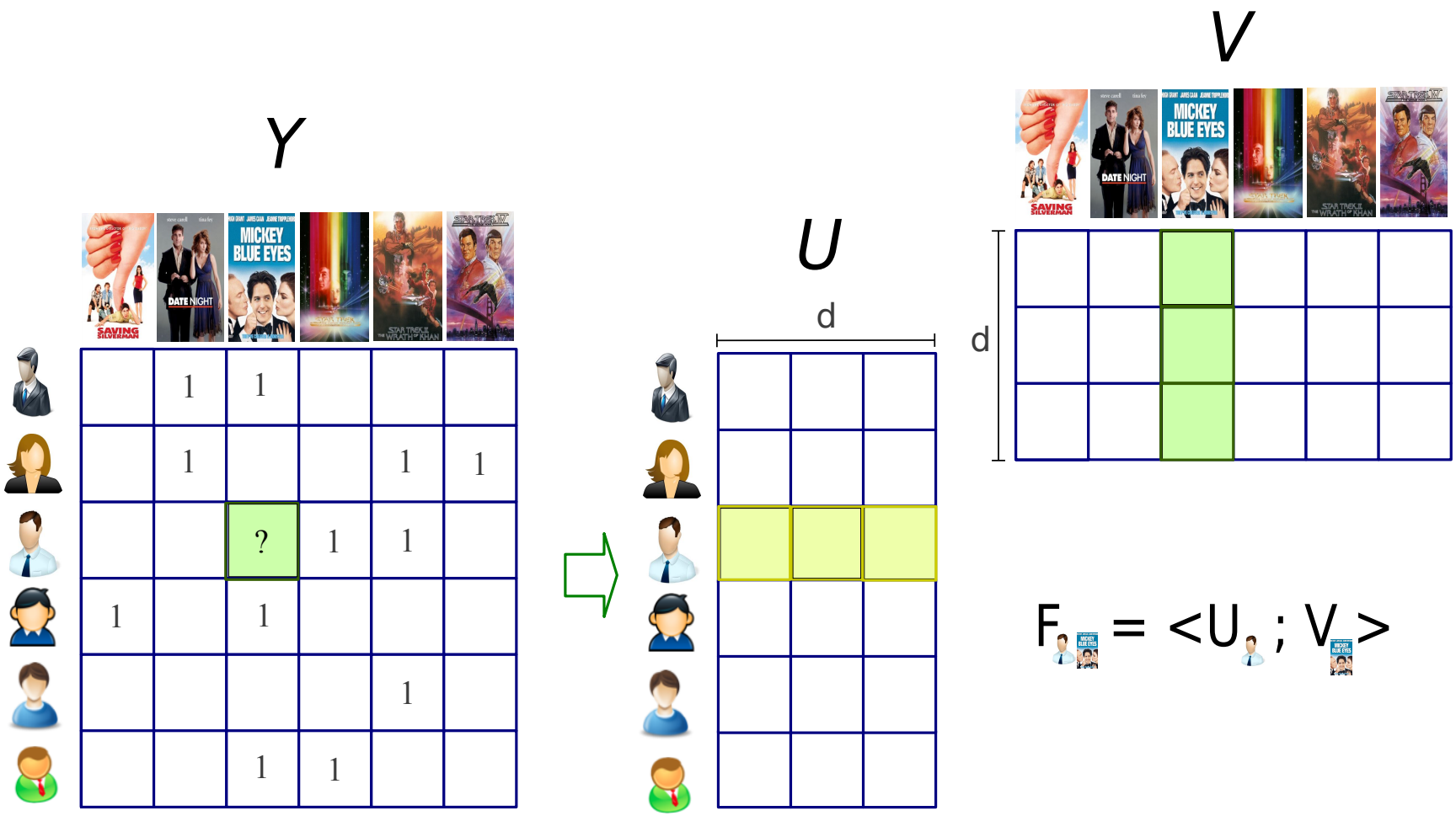
Really sparse

- 4 places per user
- 20 friends per user

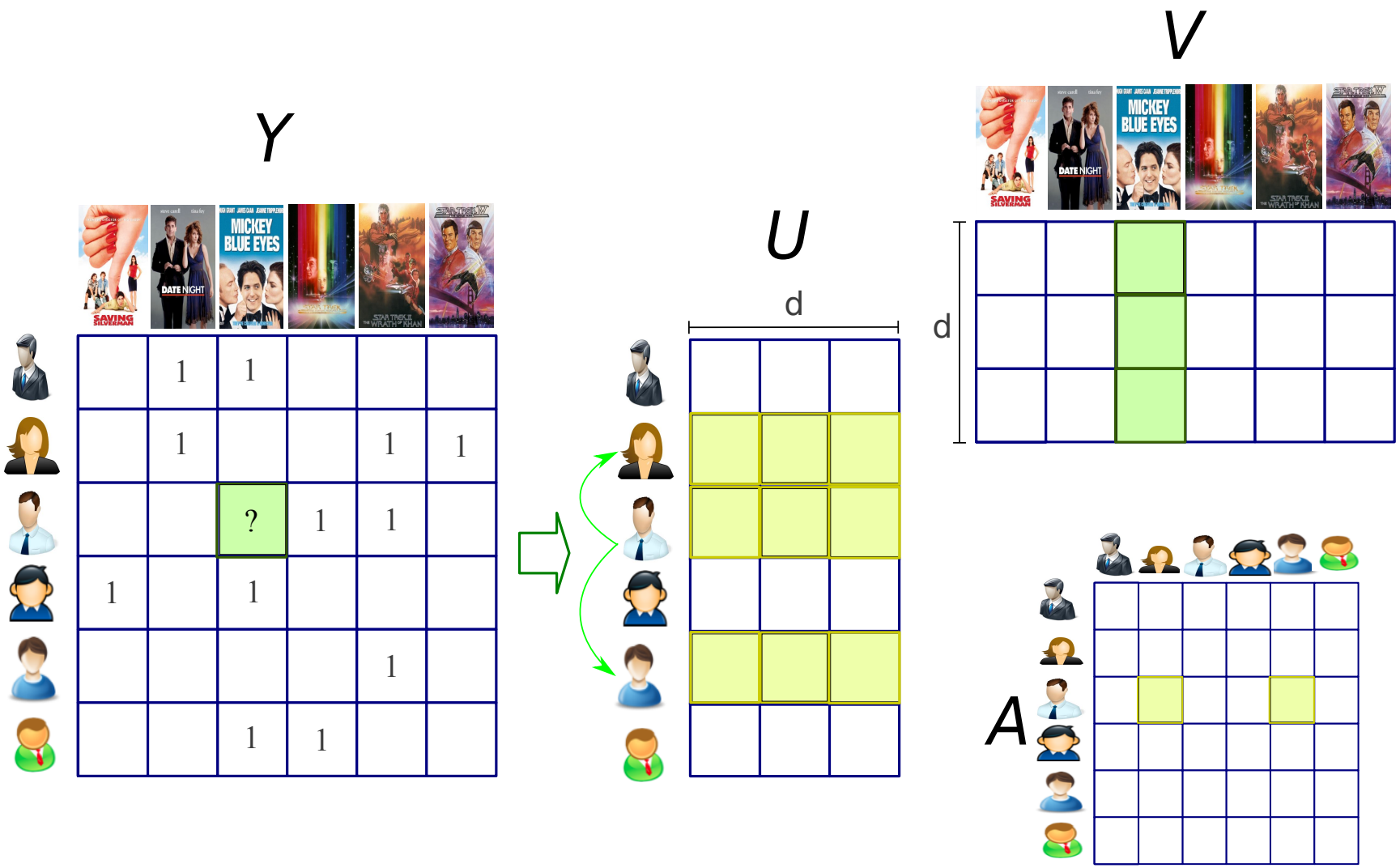




Objective



Objective



Existing approaches

Cost functions

$$\min_{U,V} \mathcal{L}_1(Y, UV) + \mu \mathcal{L}_2(A, UU^\top) + \lambda \Omega(U, V)$$ Yang *et al.* (2011)

$$\min_{U,V} \mathcal{L}(Y, UV) + \mu \sum_{i=1}^{n} \| U_i - \frac{1}{|\mathcal{F}_i|} \sum_{k \in \mathcal{F}_i} U_k \|_F^2 + \lambda \Omega(U, V)$$ Ma *et al.* (2011)

- $\mathcal{F}_i$ set of i's friends
- $A_{ii'} = 1$ if $i' \in \mathcal{F}_i$

Decision function

$$\min_{U,V} \sum_{(i,j) \in \mathcal{Y}} c_{ij} \Big(\underbrace{U_i V_j + \sum_{k \in \mathcal{F}_i} \frac{A_{ik} U_k V_j}{|\mathcal{F}_i|}}_{\hat{F}_{ij}} - Y_{ij} \Big)^2 + \Omega_{U,V,A}$$ Ma *et al.* (2009)

Decision function

Decision function

$$\hat{F}_{ij} = U_i V_j + \sum_{k \in \mathcal{F}_i} \frac{A_{ik} U_k V_j}{|\mathcal{F}_i|} \quad \text{Ma } et\ al. \text{ (2009)} \tag{1}$$

Loss function

$$\min_{U,V,A} \sum_{(i,j) \in \mathcal{Y}} c_{ij} \left(U_i V_j + \sum_{k \in \mathcal{F}_i} \frac{A_{ik} U_k V_j}{|\mathcal{F}_i|} - Y_{ij} \right)^2 + \Omega_{U,V,A} \tag{2}$$

- c_{ij} penalizing 1 more than 0.
- $\Omega_{U,V,A} = \lambda_1 \|U\|_{\text{Fro}}^2 + \lambda_2 \|V\|_{\text{Fro}}^2 + \lambda_3 \|A\|_{\text{Fro}}^2$

Contribution

Learn friendship influence

Algorithm

Alternating least square minimization – SE CoFi

Input: Y and $\mathcal{F}$

intialize U, V and A

Repeat

For each user $i \in \mathcal{U}$

update U_i

For each item $j \in \mathcal{V}$

update V_j

For each user $i \in \mathcal{U}$

For each user $k \in \mathcal{F}_i$

update A_{ik}

Until convergence

Tuenti and Epinions dataset

Epinions social network

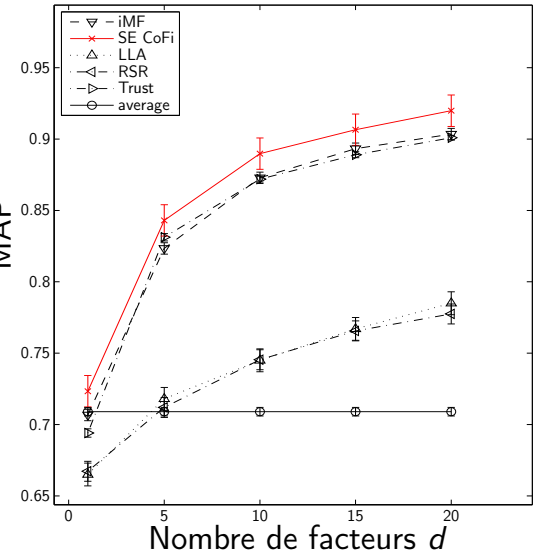
Dataset information

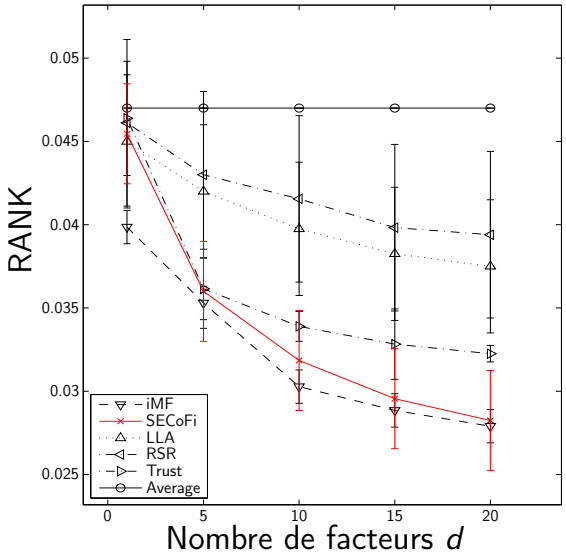
This is a who-trust-whom online social network of a a general consumer review site Epinions.com. Members of the site can decide whether to "trust" each other. All the trust relationships interact and form the Web of Trust which is then combined with review ratings to determine which reviews are shown to the user.

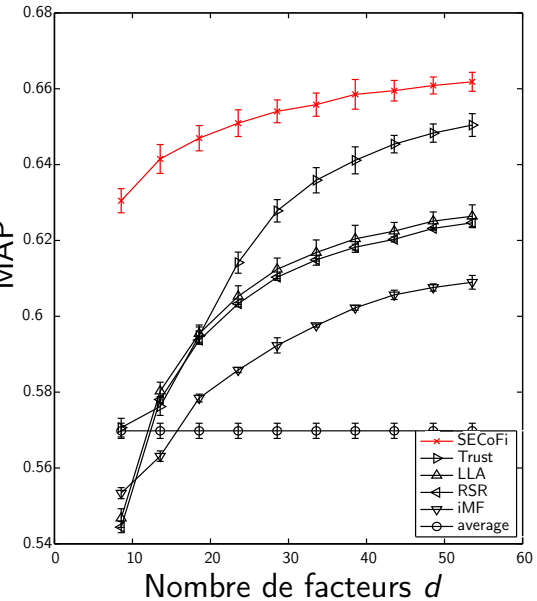
Dataset statistics	
Nodes	75879
Edges	508837
Nodes in largest WCC	75877 (1.000)
Edges in largest WCC	508836 (1.000)
Nodes in largest SCC	32223 (0.425)
Edges in largest SCC	443506 (0.872)
Average clustering coefficient	0.1378
Number of triangles	1624481
Fraction of closed triangles	0.0229
Diameter (longest shortest path)	14
90-percentile effective diameter	5

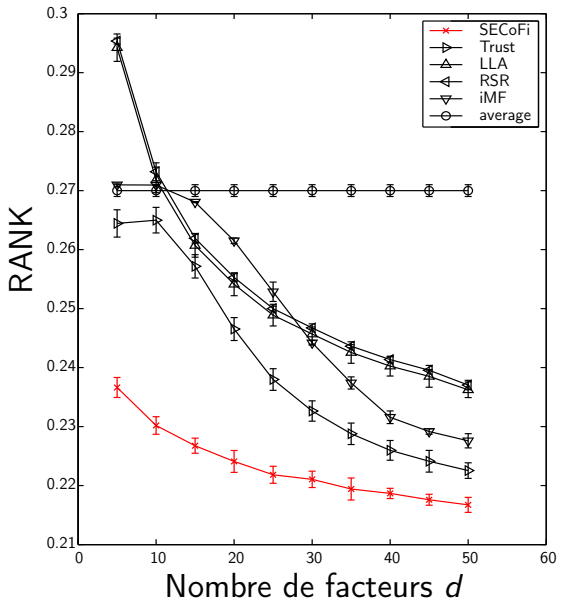
- Tuenti: 13 M users and 10 k places (20 friends)
- Epinion: 50 k users et 100 items (5 friends)

Tuenti and Epinions



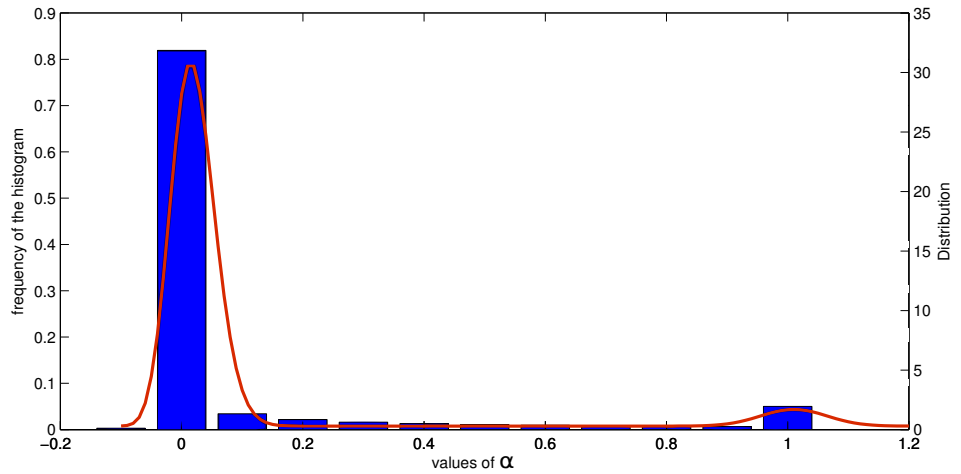




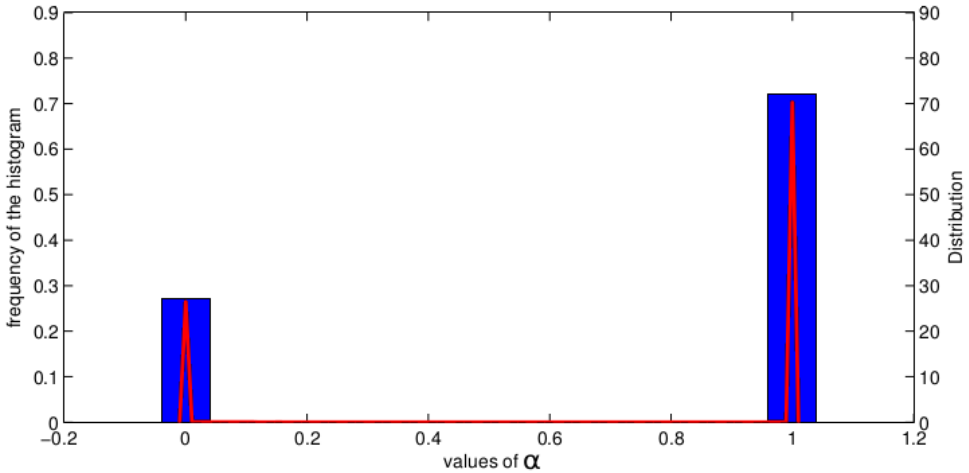


- Tuenti: 13 M users and 10 places (20 friends)
- Epinion: 50 k users et 100 items (5 friends)
- using 24 cores with 100Go

Back on friends' influence



Tuenti



Epinion

Affinity principle (*homophily effect*), no negative influence

Conclusion

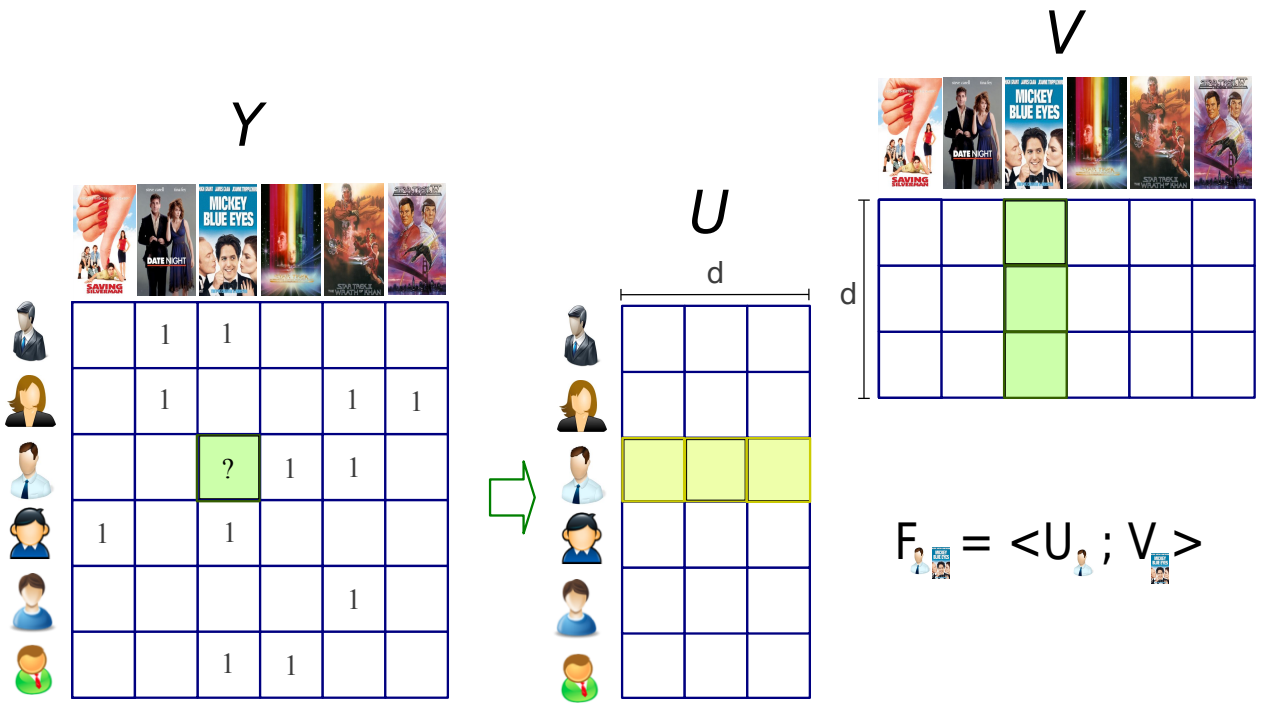
Summary

- matrix AND graph
- large scale datasets
- recommendation improvement
- influence measure

Perspectives

- take advantage of the framework
- take into account more contextual data
  - GPS, places types, ...
- towards friends recommendations (learn α)

Questions?



Matrix factorization, recommender systems, personalized with preferences