RECOMMENDER SYSTEMS

Sergey Nikolenko

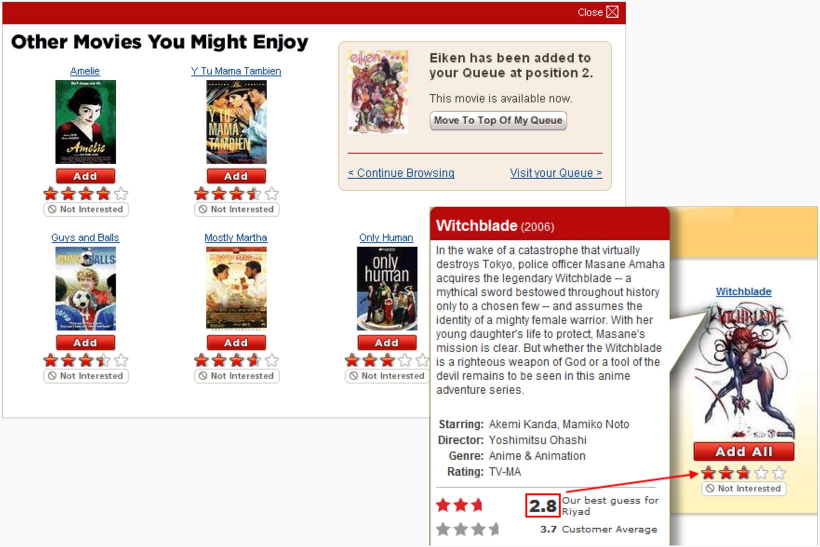
Harbour Space University, Barcelona, Spain
April 10, 2017

INTRODUCTION

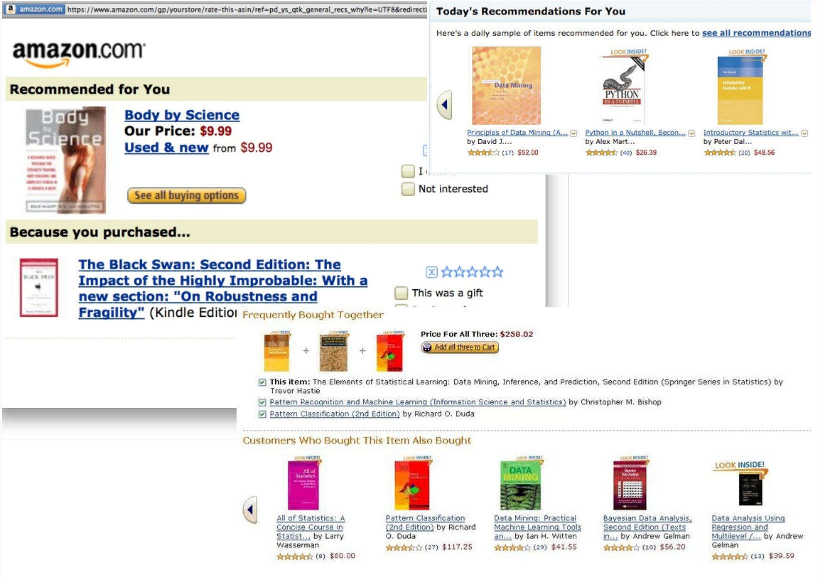
PREVIEW

- Main ideas:
  (1) classical collaborative filtering: nearest neighbors and how to scale them;
  (2) matrix decompositions: why, how, and what else;
  (3) extensions: what can we add to a recommender system; in particular, content-based recommendations;
  (4) non-personalized recommendations with emphasis on speed.

- Recommender systems analyze the users' interests and aim to predict what is most interesting for a specific user at this time.
- Leading recommender systems usually fall into one of two categories:
  (1) we "sell" some goods or services online; the users either explicitly rate the goods or simply buy something; we want to recommend an item that would most interest this user; examples: Netflix, Amazon;
  (2) we are a web portal and make money through advertising; we need to show links that the users will click: Mail.Ru, Yahoo!, Google, Yandex, content providers, news web sites.

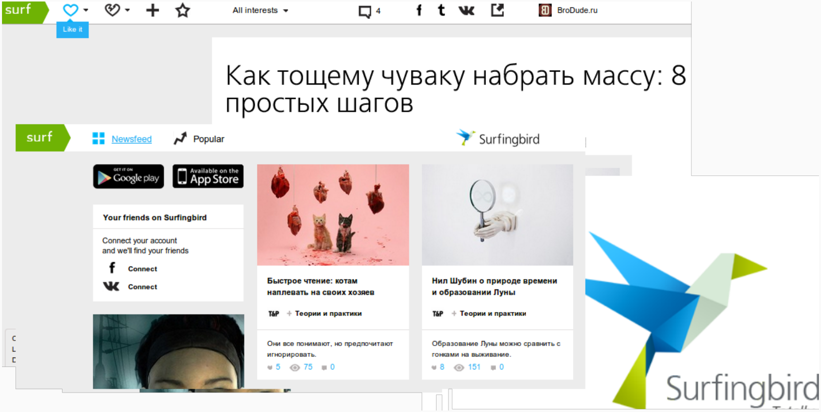

Close
Other Movies You Might Enjoy
Amelie
Add
Not Interested
Y Tu Mama Tambien
Add
Not Interested
Guys and Balls
Add
Not Interested
Mostly Martha
Add
Not Interested
Only Human
Add
Not Interested
Eiken has been added to your Queue at position 2.
This movie is available now.
Move To Top Of My Queue
< Continue Browsing
Visit your Queue >
Witchblade (2006)
In the wake of a catastrophe that virtually destroys Tokyo, police officer Masane Amaha acquires the legendary Witchblade -- a mythical sword bestowed throughout history only to a chosen few -- and assumes the identity of a mighty female warrior. With her young daughter's life to protect, Masane's mission is clear. But whether the Witchblade is a righteous weapon of God or a tool of the devil remains to be seen in this anime adventure series.
Starring: Akemi Kanda, Mamiko Noto
Director: Yoshimitsu Ohashi
Genre: Anime & Animation
Rating: TV-MA
2.8 Our best guess for Riyad
3.7 Customer Average
Witchblade
Add All
Not Interested

AMAZON



amazon.com https://www.amazon.com/gp/yourstore/rate-this-asin/ref=pd_ys_qtk_general_recs_why?ie=UTF8&redirect
amazon.com
Recommended for You
Body by Science
Our Price: $9.99
Used & new from $9.99
See all buying options
Not interested
Because you purchased...
The Black Swan: Second Edition: The Impact of the Highly Improbable: With a new section: "On Robustness and Fragility" (Kindle Editio
This was a gift
Today's Recommendations For You
Here's a daily sample of items recommended for you. Click here to see all recommendations
Principles of Data Mining (A... by David J.... (17) $52.00
Python in a Nutshell, Secon... by Alex Mart... (40) $26.39
Introductory Statistics wit... by Peter Dal... (20) $48.56
Frequently Bought Together
Price For All Three: $258.02
Add all three to Cart
This item: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics) by Trevor Hastie
Pattern Recognition and Machine Learning (Information Science and Statistics) by Christopher M. Bishop
Pattern Classification (2nd Edition) by Richard O. Duda
Customers Who Bought This Item Also Bought
All of Statistics: A Concise Course in Statist... by Larry Wasserman (8) $60.00
Pattern Classification (2nd Edition) by Richard O. Duda (27) $117.25
Data Mining: Practical Machine Learning Tools an... by Ian H. Witten (29) $41.55
Bayesian Data Analysis, Second Edition (Texts in... by Andrew Gelman (10) $56.20
Data Analysis Using Regression and Multilevel /... by Andrew Gelman (13) $39.59

SURFINGBIRD



ONLINE VS. OFFLINE

- A recommender system may have two different “levels”:
    - global estimates, slowly changing features and preferences, dependence on permanent user features (geography, demographics) etc.;
    - short-term trends, hotness, fast changes in interest.

- These are different problems with different methods, so there are two classes of models:
  - *offline models* extract global dependencies (this is usually called collaborative filtering). The purpose is to find and recommend something for a specific user, work with “long tails” of the distributions of both users and items;
  - *online models* must react very quickly, they extract short-term trends and recommend whatever is hot right now.

CLASSICAL COLLABORATIVE FILTERING

- Notation:
  - i always denotes users (N in total, $i = 1..N$);
  - a always denotes items (web sites, goods, movies...) that we recommend (M in total, $a = 1..M$);
  - when a user i rates item a, it is captured as a response (rating) $r_{i,a}$; this is a random value, of course.
- The goal is to predict estimates $r_{i,a}$ by features x_i and x_a for all elements in the dataset and some already known $r_{i',a'}$.
- We denote prediction by $\hat{r}_{i,a}$.

- Nearest neighbors: let's introduce a distance (similarity) between users and recommend to you what people similar to you have liked.
- Distance:
  - correlation coefficient (Pearson's coefficient)

$$w_{i,j} = \frac{\sum_a \left(r_{i,a} - \bar{r}_a\right)\left(r_{j,a} - \bar{r}_a\right)}{\sqrt{\sum_a \left(r_{i,a} - \bar{r}_a\right)^2}\sqrt{\sum_a \left(r_{j,a} - \bar{r}_a\right)^2}},$$

    where $\bar{r}_a$ is the average rating of item a among all users;
  - cosine of the angle between rating vectors:

$$w_{i,j} = \frac{\sum_a r_{i,a} r_{j,a}}{\sqrt{\sum_a r_{i,a}^2}\sqrt{\sum_a r_{j,a}^2}}.$$

- The simplest way to construct a prediction for a new rating $\hat{r}_{i,a}$ is the sum of ratings for other users weighted by their similarities to user i:

$$\hat{r}_{i,a} = \bar{r}_a + \frac{\sum_j \left(r_{j,a} - \bar{r}_j\right) w_{i,j}}{\sum_j |w_{i,j}|}.$$

- This is called the GroupLens algorithm, the grandfather of recommender systems.
- We can restrict the sum to nearest neighbors so that we don't have to sum over everybody:

$$\hat{r}_{i,a} = \bar{r}_a + \frac{\sum_{j \in \mathrm{knn}(i)} \left(r_{j,a} - \bar{r}_j\right) w_{i,j}}{\sum_{j \in \mathrm{knn}(i)} |w_{i,j}|}.$$

- Natural extension: let's re-weigh the items according to how often they have been rated; if something is liked by everybody it's not very useful.
- Inverse user frequency: $f_a = \log \frac{N}{N_a}$, where N is the total number of users, N_a – number of users who rated a. We get

$$w_{i,j}^{\mathrm{iuf}} = \frac{\sum_a f_a \sum_a f_a r_{i,a} r_{j,a} - \left(\sum_a f_a r_{i,a}\right)\left(\sum_a f_a r_{j,a}\right)}{\sqrt{\sum_a f_a \left(\sum_a f_a r_{i,a}^2 - \left(\sum_a f_a r_{i,a}\right)^2\right)}\sqrt{\sum_a f_a \left(\sum_a f_a r_{j,a}^2 - \left(\sum_a f_a r_{j,a}\right)^2\right.}}$$

and for the cosine:

$$w_{i,j}^{\mathrm{iuf}} = \frac{\sum_a f_a^2 r_{i,a} r_{j,a}}{\sqrt{\sum_a (f_a r_{i,a})^2}\sqrt{\sum_a (f_a r_{j,a})^2}}.$$

- Symmetrical approach – item-based collaborative filtering. Compute similarity between items, choose similar items.
- Amazon: customers who bought this item also bought...
- Can be more efficient since we can always compute item similarity offline and get new predictions for a user online.

- It's hard to find nearest neighbors algorithmically (k-d-trees don't work in large dimensions).
- Large-scale recommender systems use approximations.
- E.g., LSH (locality sensitive hashing) with min-hashing:
    - take several hash functions, compute them for every item;
    - for every user compute the minimal value of hash functions for its items;
    - look for neighbors only among those users that have identical values in at least one hash.

WHAT ABOUT LIKES?

- We have only considered explicit ratings.
- But often we only have the sets of “consumed” or “liked” items I and J for users i and j:
    - likes (usually very few dislikes);
    - bought goods without explicit ratings.
- This is called *implicit feedback*. What do we do?

- We need to define distance between sets; Jaccard similarity

$$w_{i,j} = \mathrm{Jaccard}(I, J) = \frac{|I \cap J|}{|I \cup J|}.$$

- We can introduce user weights and then use GroupLens.

- Jaccard similarity is even more popular for item-based CF.
- We define similarity between a and b via the sets of users A and B who consumed it:

$$w_{a,b} = \mathrm{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

- Often works well, but there are problems.
- E.g., what if item a is rare, b is more popular, and almost all $i \in A$ have also consumed b? (very common situation)

WHAT ABOUT LIKES?

- Jaccard similarity does not suit imbalanced cases because it is symmetric.
- Let's make it asymmetric:

$$w_{a,b} = \frac{|A \cap B|}{|A|}, \quad w_{b,a} = \frac{|A \cap B|}{|B|}.$$

- Now the previous example is fine, but there still are problems.
- What if one of the items is very popular, and everybody has seen it? *Banana trap*.

- One more variation – the method of associations:

$$w_{a,b} = \frac{|A \cap B| \,/\, |A|}{|\bar{A} \cap B| \,/\, |\bar{A}|},$$

where $\bar{A}$ is the complement of A.
- In practice it is usually easy to simply try all of this and choose what works best.

MATRIX DECOMPOSITIONS

- Let's now try to construct a model for a rating.
- What does a rating that user i gives to item a consist of?
- There are kind and harsh users, good and bad items.
- *Baseline predictors* $b_{i,a}$:

$$b_{i,a} = \mu + b_i + b_a.$$

- To find the predictors, let's make it into a probabilistic model.
- We add normally distributed noise and get the model

$$r_{i,a} \sim \mathcal{N}\left(\mu + b_i + b_a, \sigma^2\right).$$

- We can now add prior distributions and optimize

$$b_* = \arg\min_b \sum_{(i,a)} \left(r_{i,a} - \mu - b_i - b_a\right)^2 + \lambda_1 \left(\sum_i b_i^2 + \sum_a b_a^2\right).$$

- How do we train this model?

- That's just linear regression!
- Note that often ratings are binary (like/dislike).
- Then it makes more sense to use the logistic sigmoid:

$$b_{i,a} = \sigma(\mu + b_i + b_a), \quad \sigma(x) = \frac{1}{1 + e^{-x}}.$$

- And we now have logistic regression instead of linear.
- This is often a good idea even for several ratings.

- How do we personalize and train the rest of a rating?
- We have a huge $N \times M$ matrix where only some small fraction of elements are known.
- So we make assumptions on the structure of the matrix and predict the rest.

- SVD (singular value decomposition) – assume that matrix X has low rank and decompose it.
- But we can also get to the same model from the other side.
- Fix some number f of *latent factors* that define each item and the preferences of each user.
- A user is now a vector $p_i \in \mathbb{R}^f$; an item, a vector $q_a \in \mathbb{R}^f$.

- And we model the preference as a scalar product $q_a^\top p_i = \sum_{j=1}^{f} q_{a,j} p_{i,j}$.
- Adding baseline predictors, we get the following model for a rating:

$$\hat{r}_{i,a} \sim \mu + b_i + b_a + q_a^\top p_i.$$

- How do we train it?

- SGD – stochastic gradient descent.
- Compute the gradient of the likelihood function, iterate over training samples, update on every step:

$$
\begin{aligned}
b_i &:= b_i + \gamma \left(e_{i,a} - \lambda_2 b_i \right), \\
b_a &:= b_a + \gamma \left(e_{i,a} - \lambda_2 b_a \right), \\
q_{a,j} &:= q_{a,j} + \gamma \left(e_{i,a} p_{i,j} - \lambda_2 q_{i,j} \right) \text{ for all } j, \\
p_{i,j} &:= p_{i,j} + \gamma \left(e_{i,a} q_{a,j} - \lambda_2 p_{i,j} \right) \text{ for all } j,
\end{aligned}
$$

where γ is the learning rate

- ALS – alternating least squares.
- Note that if in $\hat{r}_{i,a} \sim \mu + b_i + b_a + q_a^\top p_i$ we fix p_i, we will get linear regression w.r.t. q_a, and vice versa.
- ALS is similar to EM; repear until convergence:
    - fix p_i, train q_a;
    - fix q_a, train p_i.
- Usually faster and more robust than SGD.

- The same remark about the logistic variation: for binary ratings we can consider

$$\hat{r}_{i,a} \sim \sigma(\mu + b_i + b_a + q_a^\top p_i).$$

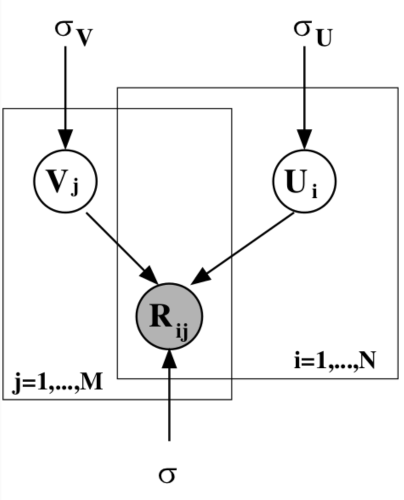
- Then the SGD will simply get the sigmoid's probabilities $\sigma'(x) = \sigma(x)(1 - \sigma(x))$.
- And in ALS instead of a linear regression we will have to train logistic regression on every iteration.

- We can also add external information to this model.
- Suppose there are extra factors y_a for the items that characterize a user based on what he has seen but not rated.
- The model is now

$$\hat{r}_{i,a} = \mu + b_i + b_a + q_a^\top \left(p_i + \frac{1}{\sqrt{|V(i)|}} \sum_{b \in V(i)} y_b \right),$$

  where $V(i)$ is the set of items that this user has seen ($\frac{1}{\sqrt{|V(i)|}}$ controls the variance).
- This is called SVD++.

- Suppose we want to decompose the rating matrix into low rank matrices
$$\hat{R} = U^\top V.$$
- The likelihood is
$$p(R \mid U, V, \sigma^2) = \prod_i \prod_a \left(\mathcal{N}(r_{i,a} \mid u_i^\top v_j, \sigma^2)\right)^{[i \text{ rated } a]}.$$
- Adding Gaussian priors on U and V, we get
$$p(U \mid \sigma_U^2) = \prod_i \mathcal{N}(U_i \mid 0, \sigma_U^2 I), \quad p(V \mid \sigma_V^2) = \prod_a \mathcal{N}(V_a \mid 0, \sigma_V^2 I).$$


σ_V
σ_U
V_j
U_i
R_{ij}
i=1,...,N
j=1,...,M
σ

- If we simply fix σ^2, σ_V^2, and σ_U^2, they will serve as regularizers, as in regular SVD.
- The difference is that we can now find optimal $\sigma = (\sigma^2, \sigma_V^2, \sigma_U^2)$ by maximizing the total likelihood of the model

$$\sigma^* = \arg\max_\sigma p(R \mid \sigma) = \arg\max_\sigma \int p(R, U, V \mid \sigma) dU dV$$

with EM:

  - first fix σ and find

$$f(\sigma) = \mathrm{E}_{U,V|R,\sigma} \left[\log p(R, U, V \mid \sigma) \right];$$

  - then maximize

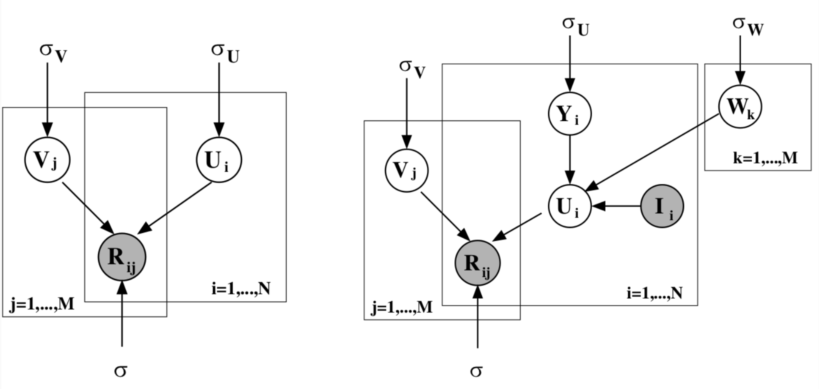
$$\sigma := \arg\max_\sigma f(\sigma).$$

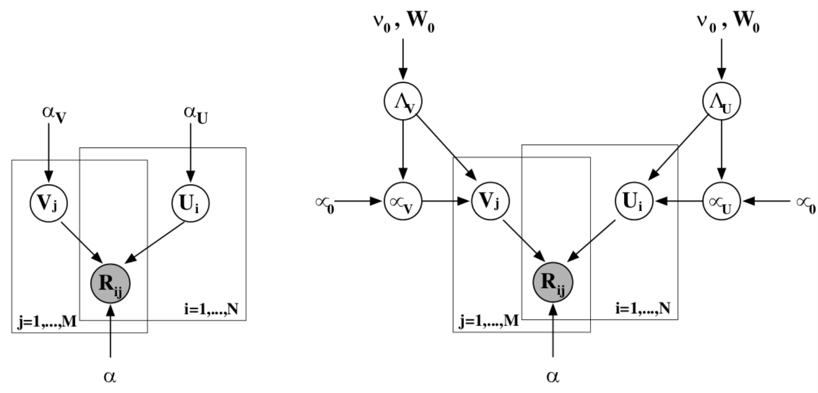
- Modification: users with few ratings in PMF will get posteriors very close to the "average user".
- To generalize better to this case, we can add factors that change the priors depending on how many and what items a user has rated:

$$U_i = Y_i + \frac{\sum_a [i \text{ rated } a] W_a}{\sum_a [i \text{ rated } a]}.$$

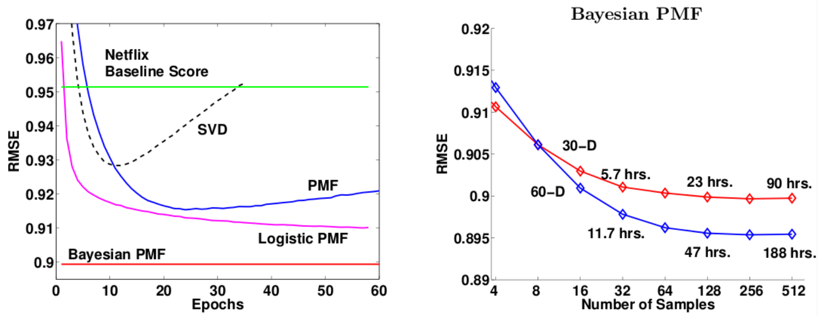
- And we get $p(W \mid \sigma_W^2) = \prod_i \mathcal{N}(W_i \mid 0, \sigma_W^2 I)$.

GRAPHICAL MODEL

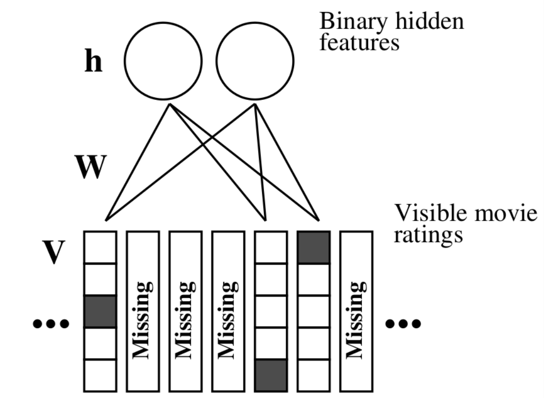




α_V
α_U
V_j
U_i
R_{ij}
j=1,...,M
i=1,...,N
α
ν_0 , W_0
ν_0 , W_0
Λ_V
Λ_U
$\propto_0$
$\propto_V$
V_j
U_i
$\propto_U$
$\propto_0$
R_{ij}
j=1,...,M
i=1,...,N
α


0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.9
RMSE
Netflix
Baseline Score
SVD
PMF
Logistic PMF
Bayesian PMF
0
10
20
30
40
50
60
Epochs
Bayesian PMF
0.92
0.915
0.91
0.905
0.9
0.895
0.89
RMSE
30-D
60-D
5.7 hrs.
23 hrs.
90 hrs.
11.7 hrs.
47 hrs.
188 hrs.
4
8
16
32
64
128
256
512
Number of Samples

- One more kind of probabilistic modeling – *restricted Boltzmann machines*.
- Undirected graphical model with two levels, visible and hidden.
- In collaborative filtering we model user preferences with RBM:



- As a result, on the hidden layer we train the user model.
- Training by contrastive divergence (approximation to max likelihood).
- RBM is not *better* than SVD, but often makes *different errors*, so a combination of these models is a big improvement.

- We have considered SVD (singular value decomposition) – decomposing matrix X into a product of low rank matrices.
- This is not the only matrix decomposition, and they are all interesting in their own way.
- PCA (principal components analysis) tries to explain as much variance in the original dataset as possible.
- But often directions to clusters in the data are not orthogonal, and PCA features are hard to interpret.

- SVD (singular decomposiion) does exactly what we need when ratings are available:
    - maximizes the likelihood for known ratings (minimizes rating prediction error);
    - works with sparse matrices (minimizes only over known ratings).
- But what do we do if there are no ratings, only the fact of use? SVD won't work...

- NMF (nonnegative matrix factorization): we still decompose as

$$X \approx UV^\top,$$

  where U is $n \times f$, V is $m \times f$, and f is much less than n and m.
- But we now require that elements U and V are nonnegative.
- The features, by the way, are often better interpretable – this is a common theme.
- NMF can be implemented with ALS, but with additional complications due to constraints.

QUALITY METRICS AND EXTENSIONS

QUALITY METRICS

- One more important topic: how do we evaluate the quality of recommendations? What is the quality metric?
- When we train SVD (maximize likelihood), we optimize the mean squared error.
- Netflix Prize, for instance, asked for the same: optimize RMSE.
- But what do we need in the real application? What do we have in the test set?

- The test set has ratings of certain items evaluated by the users.
- But the problem is to give a user new recommendations!
- We don’t have to predict all ratings, we need to find items with the largest rating.
- So in reality this is a *ranking* problem! And it’s best to take quality metrics from information retrieval, where search results are evaluated not by the RMSE of the relevance function.
- In what follows we consider the binary case (like/dislike) for simplicity.

- Classical quality metrics:
  (1) precision – number of “good” (relevant, positively ranked) items in the results divided by the total number of items in the results;
  (2) recall – number of “good” items in the results divided by the total number of “good” items in the database.
- Same problems: these parameters do not depend on the ranking, we need to know in advance how many recommendations will be needed.

- Ranking quality metrics:
  - NDCG, Normalized Discounted Commulative Gain; choose top-k recommendations (k can be larger than the necessary number) and compute

$$\mathrm{DCG}_k = \sum_{i=1}^{k} \frac{2^{\hat{r}_i} - 1}{\log_2(1+i)},$$
$$\mathrm{NDCG}_k = \frac{\mathrm{DCG}_k}{\mathrm{IDCG}_k},$$

    where $\hat{r}_i$ is our estimate of the rating of item on position i, and IDCG_k is the value of DCG_k in the ranking by true values (from the test set);
  - NDCG ranges from 0 to 1 but it's hard to interpret as probability.

- Ranking quality metrics:
  - AUC, Area Under (ROC) Curve – the probability of the event that a randomly chosen pair of items with different ratings will be ranked correctly (the higher rating will be higher in the results);
  - in the binary case there is a closed form:

$$\hat{A} = \frac{S_0 - n_0(n_0 + 1)/2}{n_0 n_1},$$

    where n_0, n_1 is the number of items that the user liked and disliked, $S_0 = \sum p_i$ is the sum of positions for the liked items in the results.

- Ranking quality metrics:
  - but simple metrics are also important because a user often looks only at the very top recommendations;
  - WTA (winner takes all) – 1 if the top recommendation is a “like” and 0 otherwise;
  - Top$k$ – share of positive ratings among top-k recommendations (Top10 is sometimes called MAP – mean average precision).

- Problem: cold start.
- If we don't know anything, there's nothing we can do.
- But usually there is some set of external features, and we can try to predict the SVD features:
    - with a simple regression over the features;
    - (usually for items) with topic modeling!

- For user features x_i and item features x_a we consider the model

$$r_{i,a} \sim \mu + b_{\text{user}}(x_i) + b_{\text{item}}(x_a) + q_a^\top p_i(t),$$

where

$$b_{\text{user}}(x_i) \sim \mathcal{N}(u(x_i), \sigma_u^2),$$
$$b_{\text{item}}(x_i) \sim \mathcal{N}(v(x_i), \sigma_v^2),$$

and as u and v we can take any kind of regression [Agarwal, Chen, 2009].

- Or with content:
  - extract topics from the items (LDA); or other features if it's not text;
  - we get a distribution $z_{a,k}$ for every a;
  - and now we train the factors $s_{i,k}$ for how much a user "likes" these topics;
  - then for a new item we estimate the topics $\hat{z}_{a,k}$ with their content and then add to the model

$$r_{i,a} \sim \ldots + \sum_k s_{i,k} \hat{z}_{a,k},$$

    which helps for cold start w.r.t. items.
- We can also train topics that specifically reflect preferences (fLDA).

- Example: let's add time, i.e., we consider user features and baseline predictors as functions of time,

$$\hat{r}_{i,a} = \mu + b_i(t) + b_a(t) + q_a^\top p_i(t),$$

where

$$\begin{aligned} b_a(t) =& b_a + b_{a,\mathrm{Bin}(t)}, \\ b_i(t) =& b_i + \alpha_i \mathrm{dev}_i(t) + b_{i,t}, \\ p_{i,f}(t) =& p_{i,f} + \alpha_{i,f} \mathrm{dev}_i(t) + p_{i,f,t} + \frac{1}{\sqrt{|V(i)|}} \sum_{b \in V(i)} y_b, \\ \mathrm{dev}_i(t) =& \mathrm{sign}(t - t_i) \left| t - t_i \right|^\beta . \end{aligned}$$

- This is called timeSVD++, one of the main components of the Netflix Prize winner.

- Suppose that users come from a social network.
- I.e., we know their friends, a part of the social graph etc.
- We can add this to the recommender model:
    - filtering/reweighting in nearest neighbors;
    - additional terms in an SVD-like decomposition;
    - decomposing the trust matrix (from the social graph) together with the matrix of ratings, change prior distribution for PMF and so on.

- Filter bubble: how do we take a user outside the usual bubble?
- Metrics that value “interesting” results:
    - diversity – make items in the list less similar;
    - novelty – choose less common items (with few ratings);
    - serendipity – choose items that are not like the user’s history.
- We only need to be able to define the similarity of items (preferably without the ratings, by content).

CARS

- CARS (context-aware recommender systems) – we recommend in a context:
  - temporal;
  - situation;
  - geographical;
  - user behaviour, and so on.

- Formally this adds new dimensions to the preference matrix.
- We get a “hypercube” of data, there are tensor decomposition methods similar to SVD.
- But simple approaches like slicing and filtering often work as well as complicated tensor decompositions...

THANK YOU!

Thank you for your attention!