2.4 RECOMMENDER SYSTEMS

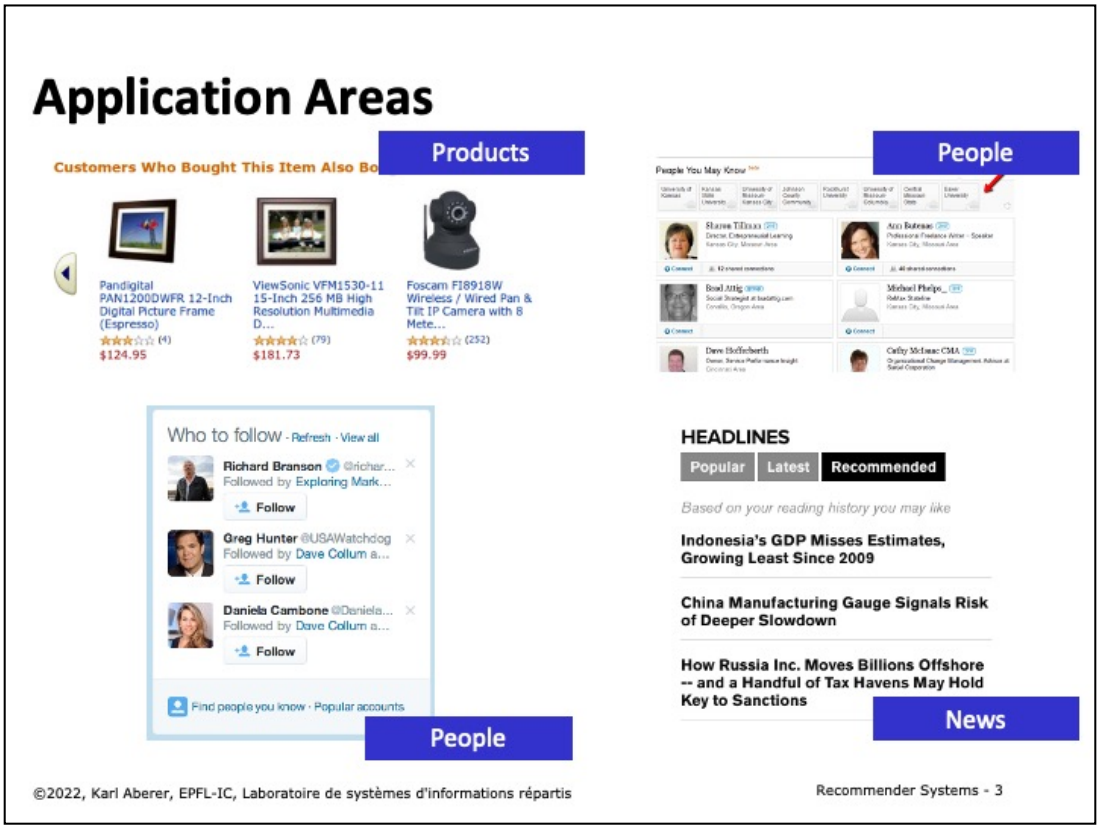
Recommender System

Given a *user* model and a set of *items*, a recommender system is a function that helps to match users with items by *ranking* the items in order of decreasing *relevance*

Learning a ranking function

Assume a user is interested in some type of items, like products, movies, books, news or jobs. A recommender system is a system that based on available data on these items proposes to the users a ranked list of items according to their preferences. Therefore recommendation can be understood as a learning problem, where the function to be learnt is the ranking of items.

The data that is available to produce such a ranking includes ratings of used of the items, demographic information on the users and characteristics of the items, including their content.



Recommender systems are widely used in many applications to provide value for the user and the provider.

For users recommender systems help, e.g., in the context of a product search,

– find things that are interesting

– narrow down the set of choices

– help explore the space of options

– Discover new things

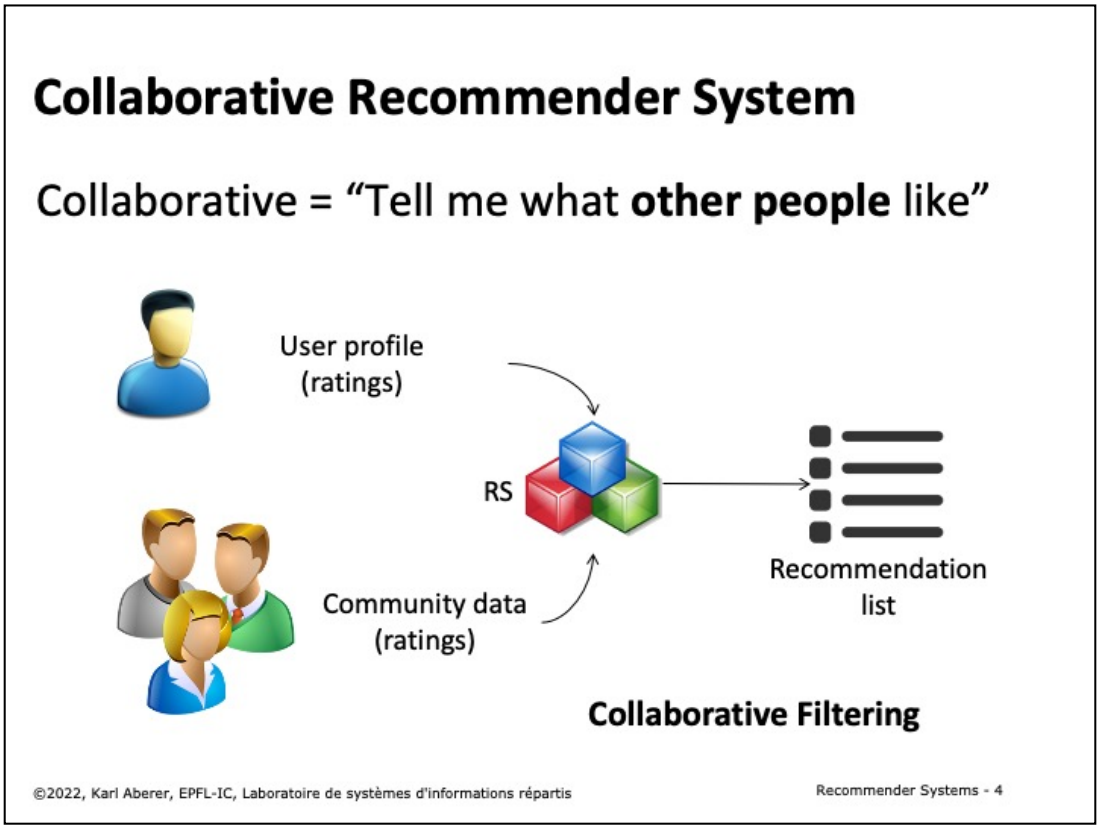
For providers recommender systems help to

– increase trust and customer loyalty with personalized services

– increase sales, click rates, page views etc.

– identify opportunities for promotion and persuasion

– obtain more knowledge about customers

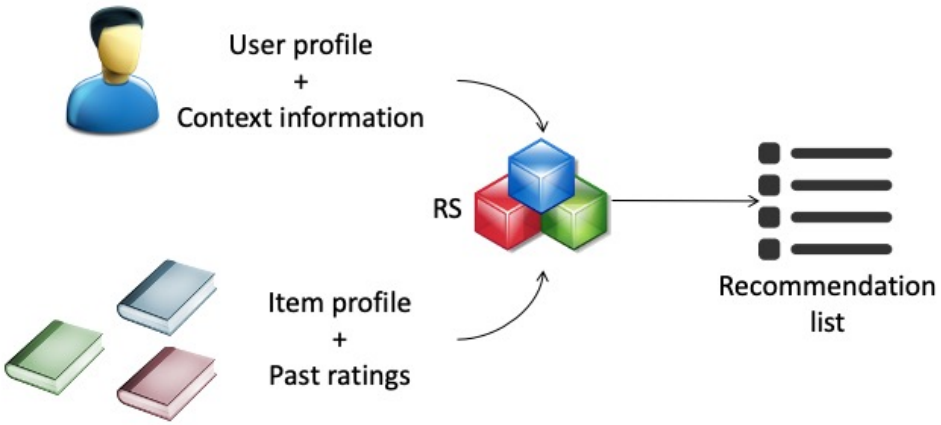


In the collaborative paradigm, the recommender system uses the user profile, primarily by the earlier ratings that are considered to represent the user preferences, and community data consisting of ratings of items by other users.

It is called collaborative because all the users participate to the recommendation by providing ratings to the items that they viewed or purchased.

Content-based Recommender System

Content-based = “Show me more of what I liked”



In the content-based paradigm, the recommender system uses the profile of the items that the user rated in the past to return a recommendation list with the most relevant items and their scores. It is called content-based because the items are associated with a set of attributes that summarize their content.

Beyond the two basic approaches of collaborative and content-based recommender systems, there exist also hybrid ones, which combine collaborative and content-based recommendation, and knowledge-based ones, which exploit domain knowledge to determine how certain item features meet the user needs.

2.4.1 Collaborative Filtering

A widely used approach to generate recommendations

- Used by large e-commerce sites
- Applicable in many domains (e.g., books, movies ...)
- Well understood and studied

Approach (*Wisdom of the crowd*)

- Users give ratings to items
- Assumption: users with similar tastes in the past will have similar tastes in the future

Collaborative recommender systems are based on collaborative filtering. In collaborative filtering, given a user and an item not yet rated by the user, the goal is to estimate the user rating for this item by looking at the ratings for the same item that were given in the past by similar users. Collaborative filtering requires a community of users that provide ratings and a way to assess user similarity.

Collaborative filtering is further distinguished into user-based collaborative filtering and item-based collaborative filtering. We will introduce these two approaches next.

User-based Collaborative Filtering

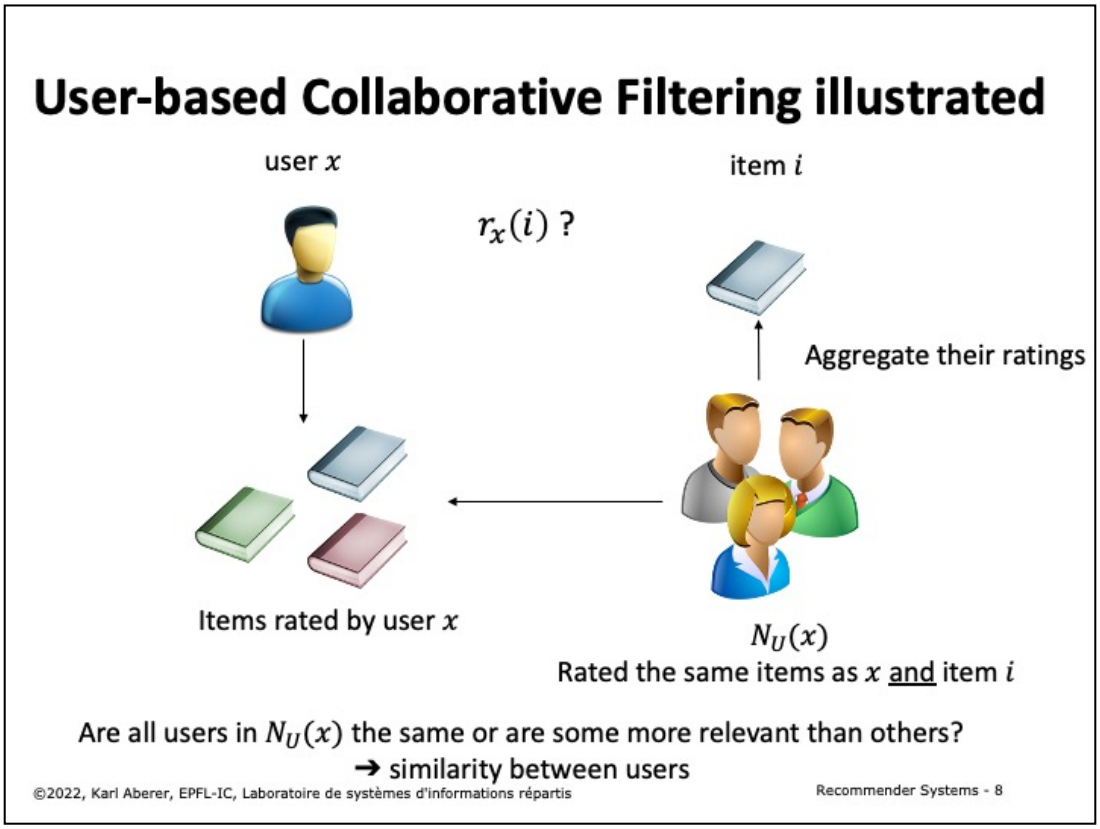
Basic technique:

Given a user x and an item i not rated by x, estimate the rating $r_x(i)$ by

1. Finding a set of users $N_U(x)$ who rated the same items as x (= neighbours of x) in the past and who have rated i
2. Aggregate the ratings of i provided by $N_U(x)$

Compute the ratings for all the items not rated by x and recommend the best-rated ones

User-based collaborative filtering is based on the idea that for an item for which a user's rating is not known, users with similar tastes are identified to estimate the rating. In order to do so the approach requires first a metric to compute similarity by users. After selecting the most similar users $N_U(x)$ it also requires a way to aggregated the ratings these users provided.



In detail user-based cllaborative filtering works as follows. First one considers all items that the user in question has already rated. Then other users are searched that have also rated these items, but also have rated the new item for which we do not know the rating of user x. In this way users with similar interests are selected.

From the ratings of those similar users of the new item then a rating of the item is estimated.

Similarity Between Users

Pearson correlation coefficient

$$sim(x,y) = \frac{\sum_{i=1}^{N}\left(r_x(i)-\bar{r_x}\right)\left(r_y(i)-\bar{r_y}\right)}{\sqrt{\sum_{i=1}^{N}\left(r_x(i)-\bar{r_x}\right)^2}\sqrt{\sum_{i=1}^{N}\left(r_y(i)-\bar{r_y}\right)^2}}$$

Cosine Similarity

$$sim(x,y) = \cos(\vartheta) = \frac{\sum_{i=1}^{N} r_x(i)\cdot r_y(i)}{\sqrt{\sum_{i=1}^{N} r_x(i)^2}\sqrt{\sum_{i=1}^{N} r_y(i)^2}}$$

x, y: users

N : number of items i rated by both x and y

$r_x(i)$: rating of user x of item i

$\bar{r_x}$: average ratings of user x

In order to determine similarity of users, a similarity measure is needed that is based on the past ratings of the users of a common set of items. We can consider those ratings as a vector and therefore an obvious choice for a similarity measure is cosine similarity returning a similarity in the interval [0.1]. For recommender systems an alternative measure that is widely used is the Pearson correlation coefficient which which returns a value between -1 and 1. Note that the Pearson correlation can be understood as a cosine similarity of the rating vectors recentered around their mean. Therefore it can adjust for different biases of users in their rating scales, i.e., users that consistently rate higher or lower.

One drawback of the Pearson correlation coefficient is that it is not defined, when the variance of one of the user ratings is 0 (e.g., a user with all ratings 1). However, in general, the correlation coefficient works well in general domains.

Example

	Item 1	Item 2	Item 3	Item 4	Item 5
U	5	3	4	4	???
User 1	3	1	2	3	3
User 2	4	3	4	3	5
User 3	3	3	1	5	4
User 4	1	5	5	2	1

$sim_{corr}(U, User1) = 0.85$
$sim_{corr}(U, User2) = 0.71$
$sim_{corr}(U, User3) = 0$
$sim_{corr}(U, User4) = -0.79$

$sim_{cos}(U, User1) = 0.97$
$sim_{cos}(U, User2) = 0.99$
$sim_{cos}(U, User3) = 0.89$
$sim_{cos}(U, User4) = 0.79$

This is an example of similarity computed using the two metrics. It is possible to see how the users are not ranked in the same way. For example, the most similar user to user U is User 1, when the Pearson correlation coefficient is used, or User 2, when the cosine similarity is used. In fact, user 1 seems to have a bias towards lower ratings, but has a similar rating pattern as user U.

Aggregate the Ratings

A common aggregation function is

$$r_x(i) = \bar{r}_x + \frac{\sum_{y \in N_U(x)} sim(x,y)(r_y(i) - \bar{r}_y)}{\sum_{y \in N_U(x)} |sim(x,y)|}$$

$N_U(x)$: neighbours of user x

i : item not rated by x

The aggregation function calculates whether the neighbours' ratings for the unseen item *i* are higher or lower than their average. We can consider this term as the bias of the user *y* w.r.t. item *i*. Then the function combines the rating bias using the similarity as a weight, so that the most similar neighbours will have more importance. Finally, the aggregated neighbours' bias is added/subtracted from user's *x* average rating.

Example

	Item 1	Item 2	Item 3	Item 4	Item 5
U	5	3	4	4	???
User 1	3	1	2	3	3
User 2	4	3	4	3	5
User 3	3	3	1	5	4
User 4	1	5	5	2	1

Closest users to U: sim_{corr}(U, User1) = 0.85, sim_{corr} (U, User2) = 0.71

$$\bar{r}_U = 4$$

$$(r_{U1}(I5) - \bar{r}_{U1}) = 3 - \frac{12}{5} = \frac{3}{5}, (r_{U2}(I5) - \bar{r}_{U2}) = 5 - \frac{19}{5} = \frac{6}{5}$$

$$r_U(I5) = 4 + \frac{0.85 * \frac{3}{5} + 0.71 * \frac{6}{5}}{0.85 + 0.71} = 4.87 \approx 5$$

Here we demonstrate in detail the calculation of an estimated rating with user-based collaborative filtering.

User-based Collaborative Filtering

Problems

- Cold start: users or items without ratings
- Scalability: large numbers of users
- Data dispersion: highly variable ratings, difficult to find similar users

Possible solution

- Item-based collaborative filtering

A typical problem of user-based collaborative filtering is the cold-start problem. The recommendation for a new user that did not rate anything yet or for an item that was never rated by any user. Also, user-based collaborative filtering is problematic when there are millions of users and/or items, such as we find them on the typical Internet platforms like Amazon or Netflix. For example, for each user the most similar users have to be found doing nearest neighbour search from a large set of users. With large numbers of ratings, it becomes hard to find common ratings between users, due to the high dispersion of data.

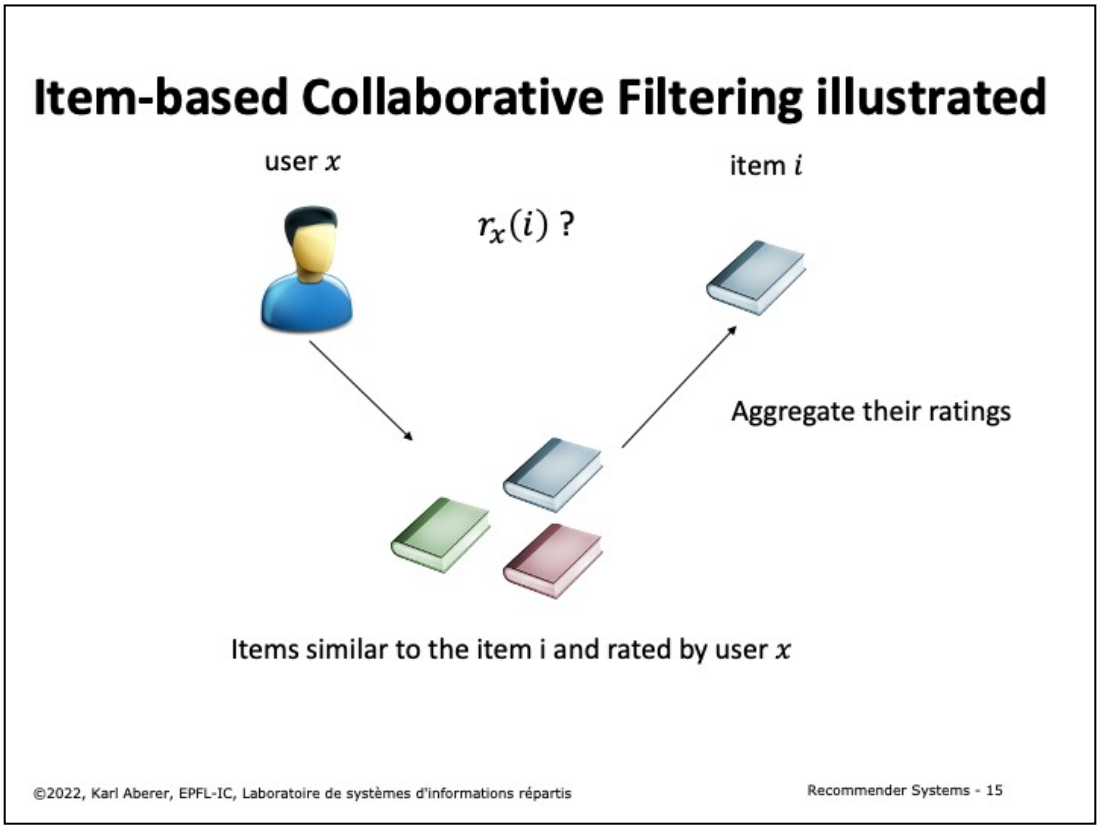
Item-based Collaborative Filtering

The basic technique: use the similarity between items (and not users) to make predictions

- Given a user x and an item i not rated by x, estimate the rating $r_x(i)$ by

    1. Find a set of items $N_I(i)$ which are similar to i and that have been rated by x
    2. Aggregate the ratings of the items in $N_I(i)$ and use the aggregation as prediction of $r_x(i)$

As opposed to user-based collaborative filtering, in item-based collaborative filtering the rating for an item for which a user's rating is not known, is derived from items that have similar ratings to the item in question, and not from users that have similar tastes as the user. Analogously, this requires a method to compute similarity among the ratings of items in order to determine a neighbourhood of similar items and a method for aggregating the ratings of the similar items.



The figure illustrates the approach. For obtaining an estimation of the rating of user x of an item i the user has not rated, first a set of items is identified that is similar, in the sense that it has received similar ratings as the item in question.

Similarity Between Items

	Item 1	Item 2	Item 3	Item 4	Item 5
U	5	3	4	4	???
User 1	3	1	2	3	3
User 2	4	3	4	3	5
User 3	3	3	1	5	4
User 4	1	5	5	2	1

sim_{corr}(item5, item1) = 0.97
sim_{corr}(item5, item2) = -0.48
sim_{corr}(item5, item3) = -0.43
sim_{corr}(item5, item4) = 0.58

The attributes that define an item are the ratings of the users different than U. The similarity can be computed with the same functions used before for user-based collaborative filtering, Pearson correlation coefficient or cosine similarity. In this example, similarity is computed with the correlation coefficient, and it is possible to see how item 1 and 4 are the most similar to item 5. Thus, aggregating the ratings of these items given by user U is an estimate of the rating of user U for item 5.

Aggregate the Ratings

A common aggregation function is

$$r_x(i) = \frac{\sum_{j \in N_I(i)} sim(i,j) r_x(j)}{\sum_{j \in N_I(i)} |sim(i,j)|}$$

$N_I(i)$: neighbours of item i

j : item rated by x

The aggregation function computes the prediction of an item i for a user x by computing the sum of the ratings given by the user on the items similar to i. Each rating is weighted by the corresponding similarity sim(i,j) between items i and j. Note that here bias in the ratings of other users is not taken into account.

Example

	Item 1	Item 2	Item 3	Item 4	Item 5
U	5	3	4	4	???
User 1	3	1	2	3	3
User 2	4	3	4	3	5
User 3	3	3	1	5	4
User 4	1	5	5	2	1

Closest items: sim_{corr}(item5, item1) = 0.97, sim_{corr}(item5, item4) = 0.58

$r_U(item5) = (0.97*5 + 0.58*4)/(0.97 + 0.58) = 4.62 \approx 5$

Here we demonstrate in detail the calculation of an estimated rating with item-based collaborative filtering.

Item-based Collaborative Filtering

Item-based collaborative filtering does not solve the cold-start problem but has better scalability

- Calculate all the pair-wise item similarities in advance
- Items similarities are more stable
- The neighbourhood $N_I(i)$ to be considered at runtime is small

Although item-based collaborative filtering does not solve the cold-start problem, it is usually more suited to tackle big datasets. In fact, the item similarities can be computed in advance, given that the items are more stable than the users. In general, particularly in product recommendation systems, new items appear at a slower pace than new users. Furthermore, the size of the neighbourhood of a given item a is in general smaller than the neighbourhood of a user x, because $N_I(i)$ is composed of the other items rated by user x (e.g., tens of DVDs bought on Amazon), while $N_U(x)$ is composed by the other users that rated the same items as x (e.g., millions of users that watched The Godfather on Netflix).

Given 3 users with ratings…

u1:	1	3
u2:	2	4
u3:	1	4

A. $Sim_{corr}(u1, u2) > Sim_{corr}(u1, u3)$
B. $Sim_{corr}(u1, u2) = Sim_{corr}(u1, u3)$
C. $Sim_{corr}(u1, u2) < Sim_{corr}(u1, u3)$

Item-based collaborative filtering addresses better the cold-start problem because …

A. usually there are fewer items than users
B. it uses ratings from items with similar content
C. item similarities can be pre-computed
D. none of the above

2.4.2 Content-based Recommendation

Collaborative filtering uses only ratings, it does not exploit any other information about the items

However, the *content* of the item might provide some useful information

- E.g., recommend sci-fi novels to users who liked Asimov books in the past

Both recommendation techniques we saw before are based on the ratings of the user exclusively, and exploit no information about the item. However, the content of the item might provide useful information for recommendation. Content-based recommendation takes into consideration the items that have been rated by the user and the content of all existing items, significantly reducing the scalability problems the recommendation is not relying on data from a large community of users.

Content-based Recommendation

The basic technique

- Given the items that have been rated by user x in the past

    1. Find the items that are similar to the past items
    2. Aggregate the ratings of the most similar items

The basic technique for content-based recommendation needs to define

1 A way to formalize the item content

2 A similarity measure between items

3 An aggregation function for the ratings

Content-based Recommendation

Most methods originate from information retrieval to extract the content of an item

Given an item i and a term t (e.g. movie synopsis, book review), compute the tf-idf weights

$$w(t,i) = tf(t,i)idf(t) = \frac{freq(t,i)}{\max_{s \in V} freq(s,i)} \log \frac{N}{n(t)}$$

N : number of items to recommend

$n(t)$: number of items where term t appears

A standard approach for content-based recommendation for items is the characterisation of the items in terms of tf-idf. Each item can be described by a set of terms that appear in their textual description description, with their associated weight, the tf-idf measure.

To compute the tf-idf measure measure the standard processing steps from information retrieval, including stopword elimination, and stemming, can be applied.

Similarity between Items

Cosine similarity

$$sim(i,j) = \cos(\theta) = \frac{\sum_{t \in V} w(t,i) w(t,j)}{\sqrt{\sum_{t \in V} w(t,i)^2} \sqrt{\sum_{t \in V} w(t,j)^2}}$$

i, j: items

V : vocabulary of all terms

$w(t,i)$: tf-idf of term t in item i

The cosine similarity in content-based recommendation considers the items i and j as two |V|-dimensional vectors, where each dimension is the tf-idf measure of a specific term t.

Making Predictions

Given a set of items D that have been rated by the user, find the nearest neighbours $N_I(i)$ in D of a new item i

Use the ratings of the nearest neighbours to predict the rating of i with the aggregation function

$$r_x(i) = \frac{\sum_{j \in N_I(i)} sim(i,j) r_x(j)}{\sum_{j \in N_I(i)} |sim(i,j)|}$$

After representing items by their tf-idf vectors, to estimate the rating of an item i not yet seen/rated by user x, the nearest neighbours of i in the subset of items that have been already rated by the user x are found and their ratings are aggregated. Note that this approach to predict the rating is analogous to the use of the kNN nearest-neighbour classification for document classification. In both cases, the nearest documents are used as predictors for the most likely label, respectively rating.

Content-based Recommendation

Problems

- Cold start problem for users with no ratings
- Content can be difficult or impossible to extract (multimedia)
- Tends to recommend “more of the same”

Scalable: tf-idf of items can be computed offline

Content-based recommendation also suffers from the cold start problem as collaborative filtering, but only for the users that did not rate anything in the past. Another problem is that sometimes the information available to extract the content is limited (e.g., movie synopsis versus full plot) or impossible to process (e.g. a sound or a video). Finally, content-based recommendation tends to recommend more of the same, it does not surprise the user with new interesting items.

On the other hand, the approach is in general more scalable, because it does not rely on a community of users to provide recommendations. Furthermore, the tf-idf measure can be computed once a new item enters the system, and then an update of the pair-wise similarities with all the existing items can be performed.

For a user that has not done any ratings, which method can make a prediction?

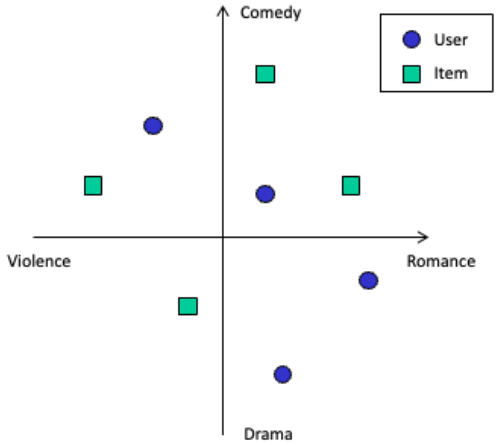
A. User-based collaborative RS
B. Item-based collaborative RS
C. Content-based RS
D. None of the above

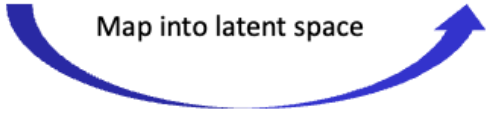
For an item that has not received any ratings, which method can make a prediction?

A. User-based collaborative RS
B. Item-based collaborative RS
C. Content-based RS
D. None of the above

2.4.3 Matrix Factorization

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





Matrix factorization transforms the user-item rating matrix into a latent factor representation, where each user and item is described as a vector in a k-dimensional space, with a dimension k significantly smaller than the number of users or items. It can be understood as a combination of user-based and item-based collaborative filtering.

The factors describe different characteristics of users, some of which might be interpretable. Proximity in the latent space between items and users indicates that the users has a preference for the corresponding item. The preference can be computed as the cosine similarity between the vectors representing the item and users.

This approach is comparable to the use of singular value decomposition for document retrieval. However, a direct application of SVD is not possible because the user-item matrix is not complete, due to missing ratings. Therefore other approaches for determining the latent space representation are required.

Derive Latent Factors

Rating Matrix R with dimension $|U|\times|I|$

- Users U and Items I

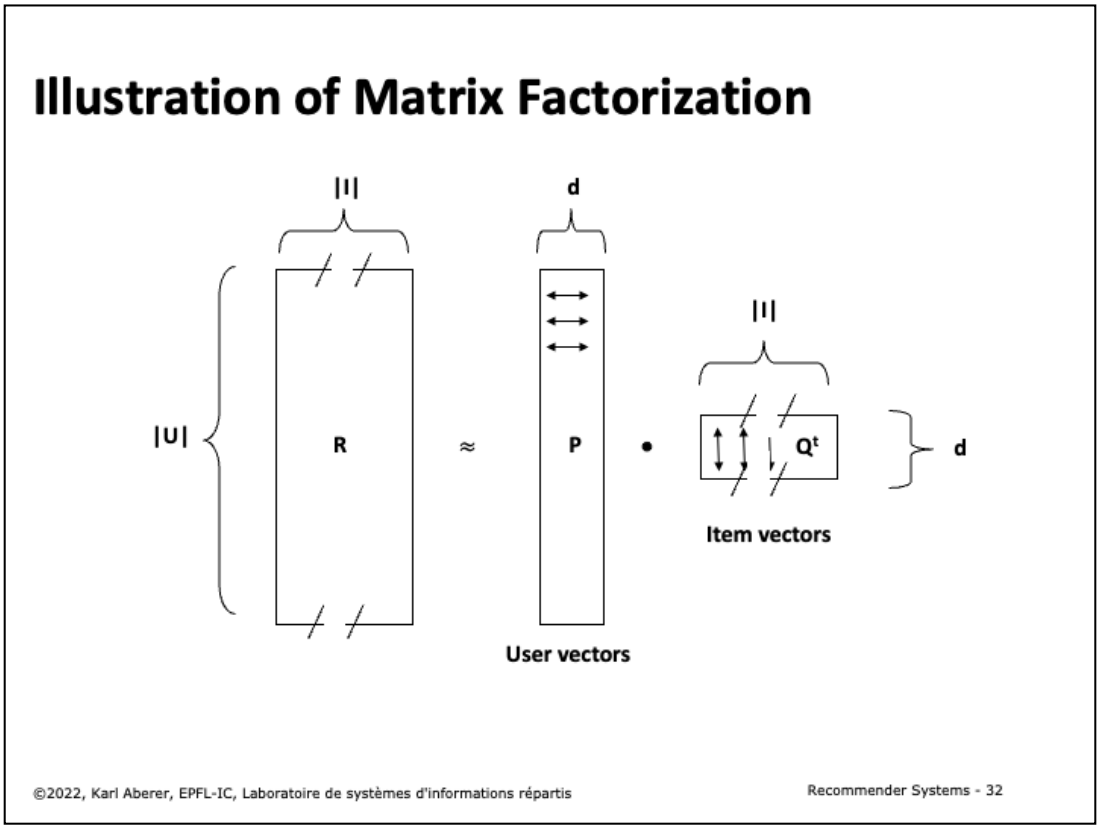
Goal: Decompose R (approximatively)

$$R \approx P\times Q^t = \hat{R}$$

d latent factors (low dimension)

- P is a $|U|\times d$ matrix: user features
- Q is a $|I|\times d$ matrix: item features

Since a algebraic approach for matrix factorization, as with SVD, is not possible, an alternative way is to produce the matrix factorization by approximation. For doing so, two factor matrices P and Q are introduced that represent the users and items in the latent space. The objective is to find these matrices, so that their produce is close, but not identical, to the original rating matrix R.



Using our approach to illustrate LSI, we can also ilustrate matrix factorization in the same way.

Computing Matrix Factorization

Problem: R has many undefined values (missing ratings)!

- SVD matrix decomposition (as used for LSI) not applicable

Formulate an optimization problem

$$\min_{p,q} \sum_{(i,j)known} \left(r_{ij} - \hat{r}_{ij}\right)^2, \hat{r}_{ij} = \sum_{k=1}^{d} p_{ik} q_{kj}$$

Apply SGD

Since SVD can not be applied to a matrix with missing values, we formulate an optimization problem. Thus, to learn the factor vectors q_i and p_u , the optimization minimizes the squared error of the differences between the entries in the original rating matrix R, and the matrix $P \times Q^t = \hat{R}$, but only for those entries in R that are defined. Note that in $\hat{R}$ there can be non-zero entries in the positions where no ratings exist.

To solve the optimization standard SGD can be applied.

Approximation Error

For a given rating r_{ij}

$$e_{ij}^2 = \left(r_{ij} - \hat{r}_{ij}\right)^2 = \left(r_{ij} - \sum_{k=1}^{d} p_{ik} q_{kj} \right)^2$$

Minimizing error: compute gradient

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2\left(r_{ij} - \hat{r}_{ij}\right) q_{kj} = -2 e_{ij} q_{kj}$$

$$\frac{\partial}{\partial q_{kj}} e_{ij}^2 = -2\left(r_{ij} - \hat{r}_{ij}\right) p_{ik} = -2 e_{ij} p_{ik}$$

To perform SGD the model is updated for each error term separately. Therefore, we can compute the gradients of the error terms independently, which results in simple linear terms.

Stochastic Gradient Descent

Update rule: for some random pair i, j

$$p_{ik} := p_{ik} - \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij} q_{kj}$$

$$q_{kj} := q_{kj} - \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij}\, p_{ik}$$

Repeat until error is small

Needs only to be computed for ratings r_{ij} that exist!

With the gradients we can then define the update rules. Since we perform SGD, we need only to update error values for which a corresponding rating exists. As a side effect, once the decomposition is computed, we obtain also estimates for the ratings for the other user-item pairs, for which not rating exists in the training data.

Regularization

In order to avoid overfitting

$$e_{ij}^2 = \left(r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj} \right)^2 + \lambda(\|P\|^2 + \|Q\|^2)$$

And then

$$p_{ik} := p_{ik} - \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha(e_{ij} q_{kj} - \lambda p_{ik})$$

$$q_{kj} := q_{kj} - \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha(e_{ij}\, p_{ik} - \lambda q_{kj})$$

In order to avoid over-fitting, as usual a regularization term is added, and the update rule is modified correspondingly. The regularization term assures that the parameters of the model are minimized.

Issues with Basic Matrix Factorization

Matrix P grows with number of users

- Problematic with large user populations, e.g., social networks
- Users modelled individually
  - Many models to manage
- Training becomes expensive
  - Large training matrix

The main drawback of matrix factorization is that it does not scale with the number of users, which in real world scenarios can become very large (e.g., customers of an ecommerce or streaming service) and therefore the training can become expensive. This probalm is addressed by the next approach that we will introduce.

Question

How does matrix factorization address the issue of missing ratings?

1. It uses regularization of the rating matrix
2. It performs gradient descent only for existing ratings
3. It sets missing ratings to zero
4. It maps ratings into a lower-dimensional space

Question

With matrix factorization one estimates ratings of unrated items

1. By retrieving the corresponding item from a user vector
2. By retrieving the corresponding item from an item vector
3. By computing the product of a user and an item vector
4. By looking up the rating in an approximation of the original rating matrix

2.4.4 SLIM, Sparse Linear Methods

Idea: model item-item relationship directly, without explicitly representing users

- Items with similar ratings have similar representation
- Compute their ratings as linear combination from a few samples -> sparse
- Avoids construction of large user-item matrix

In order to overcome the issue of scaling with the user population, alternative models have been considered. SLIM is a representative of those.

It is based on the idea to a create a representation of items, such that items with similar ratings have a similar representation. Thus, it is comparable to the approach of item-based collaborative recommender algorithms, that we have introduced earlier. The difference is that instead of comparing items by their full rating vector, consisting of the ratings of the full user population, a sparse representation of item vectors is derived. This sparse representation that is encoded in a weight aggregation matrix W allows to recover the rating of an item from (a few) ratings from similar items.

Example

R

	Item 1	Item 2	Item 3	Item 4	Item 5
U	5	3	4	4	5
User 1	3	1	2	3	3
User 2	4	3	4	3	5
User 3	3	3	1	5	4
User 4	1	5	5	2	1

Optimal vector to recover original rating

Id 5
0
0
0
0
1

Sparse vector to recover original rating

w 5
0.6
0
0
0.4
0

Recovering approximate ratings for item 5 using w_5

U: $0.6*5 + 0.4*4 = 4.6$

User 1: $0.6*3 + 0.4*3 = 3$

User 2: $0.6*4 + 0.4*3 = 3.6$ …

In this example we illustrate the idea: In item-based collaborative filtering we reconstructed an unknown rating (e.g., item 5 of user U) by considering the ratings of the most similar users for item 5.

With sparse approximation, we try to find for each item a linear combination of other items that allow to reconstruct the ratings of the item for each user In our example we would use items 1 and 4 to reconstruct the rating of item 5. This vector used for reconstructing item 5 cannot contain a 1 on the diagonal, otherwise it would be trivial to reconstruct the rating of item 5.

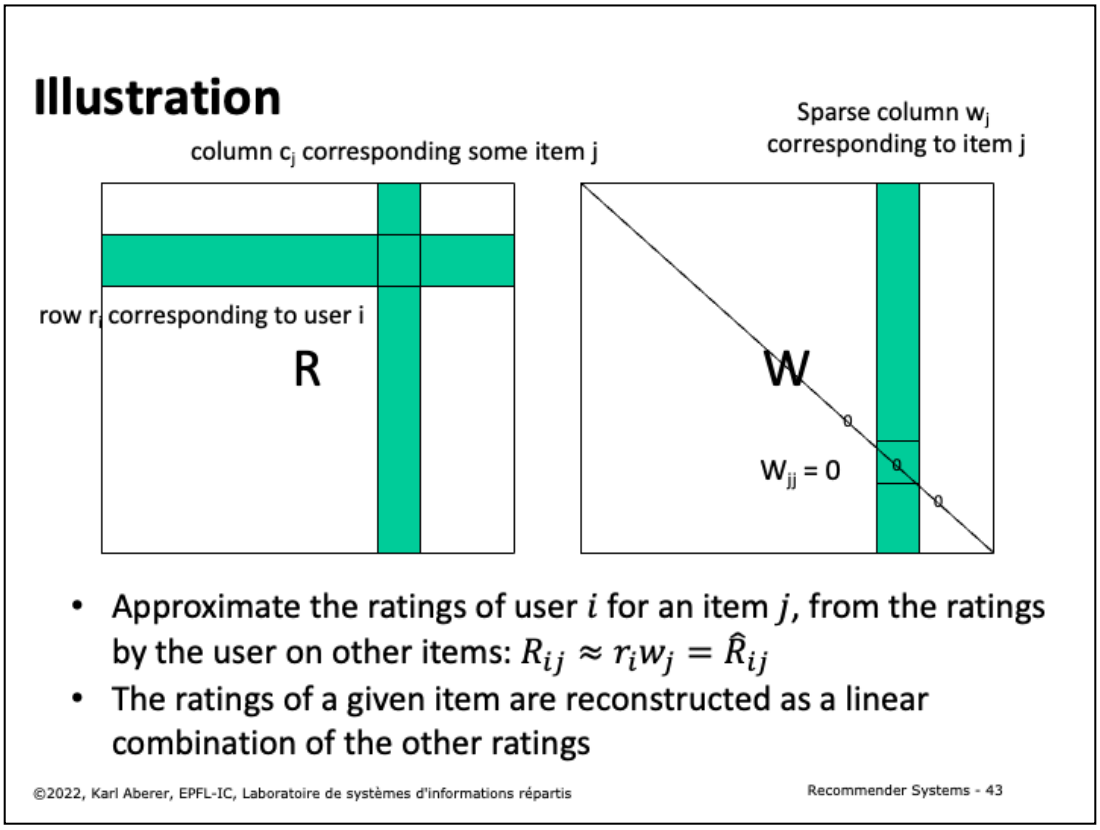
Weight Aggregation Matrix

Approximate the Rating Matrix R with dimension $|U|\times|I|$ using a **sparse** weight aggregation matrix W

$$\hat{R} \approx RW$$

- W is a $|I|\times|I|$matrix
- $\text{diag}(W) = 0$, otherwise, trivial solution with W as identity matrix

To formalize the idea, we introduct weight aggregation matrix, which represents the aggregation vectors for all items. It has the dimension $|I|\times|I|$ since it maps the ratings of items into ratings of items, and it is designed to approximate the original rating matrix R. As an additional condition it has 0 on the diagonal, otherwise it would be trivial to reconstruct the original rating matrix.



What approximating the rating for a given user i, the item vector i (which contains the ratings of the user) is multiplied with the corresponding column form W.

Recommendation with SLIM

For all items of the user without rating, compute the estimated rating score

- Since the diagonal in W is zero, the score can be reconstructed from entries for other items
- Since W is sparse only a low number of other item ratings will be used

Recommendation is done by sorting the items of user i by the estimated rating and selecting the top-k items

For performing a recommendation for a given user, for all unrated items of the user an estimated rating is computed by multiplying the users item rating vector with the sparse vectors from W corresponding to the unrated items. Since a user has in general rated only few items, and the vectors from are sparse, this is computationally more efficient than performing a full matrix multiplication. The final recommendation is obatined by selecting the top-k rated items.

Optimizing the Weight Aggregation Matrix

Impose constraints on W

- $\mathrm{diag}(W) = 0$: Ratings of an item needs to be reconstructed from other ratings
- $W \geq 0$: all entries are positive
- Minimize the norm of W (regularization)
  - Minimize $|W|_1$, minimizes number of positive weights (promotes sparsity, Lasso regression)
  - Minimize $|W|_2$, minimizes the size of weights (reduces model complexity, Ridge regression)

It remains to be seen of how the aggregation matrix can be obtained. Apart from minimizing the distance from the actual weights in R and the reconstructed weights in RW, an important aspect is to promote sparsity of the aggregation matrix. For this, standard methods from machine learning are applied. Including into the loss function the 1-norm promotes the sparsity of the matrix and including the 2-norm minimizes the sizes of the weights.

Optimization Problem

Solve

$$\min_W \frac{1}{2}|R - RW|_2^2 + \frac{\beta}{2}|W|_2^2 + \lambda|W|_1$$

subject to $\text{diag}(W) = 0$ and $W \geq 0$

As a result, this is the optimization problem to be solved in order to obtain the sparse matrix W.

Solving the Problem

Can be solved for each item separately (in parallel)

$$\min_{w_j} \frac{1}{2}\left|c_j - Rw_j\right|_2^2 + \frac{\beta}{2}\left|w_j\right|_2^2 + \lambda\left|w_j\right|_1$$

subject to $w_j \geq 0$ and $w_{j,j} = 0$

Standard methods for solving this problem exist (coordinate descent)

We observe that the optimization problem can be solved independently for all items, therefore the optimization can be performed in parallel. To solve the individual optimizations for items standard optimization techniques can be applied such as coordinate descent. In coordinate descent the function is iteratively optimized using gradient descent along the different coordinates.

Properties

- W directly represents relations among items
- Due to sparsity of matrix W top-k recommendations are less expensive to compute
  - scales with number of non-zero values
  - typically, 1% non-zero entries
- Training can be performed with a subset of users
- Users need not to be individually modelled, only their rating history needs to be stored

The SLIM method allows to compute recommendations very efficiently, as the weight aggregation matrix is very sparse. In experiments typical values have shown the the matrix has only 1% of non-zero values. To further optimize the approach, the optimization for obtaining the aggregation matrix can be performed using only a subset users, if the user population is very large. No model of users is built with this approach, only their rating history needs to be recorded.

Question

Which of the following matrices is not sparse?

1. The matrix W
2. The matrix $\hat{R}$
3. The matrix RW
4. More than one of the above are sparse

Question

If users rate about 1% of items, and W has sparsity 1%, how many multiplications are needed to recommend the top-k elements?

1. $|U| * |I| * 0.0001$
2. $|U| * k * 0.01$
3. $|U| * |I| * k * 0.0001$
4. $|U| * |I| * 0.01$

2.4.5 Evaluation of Recommender System

Standard error metrics used:

Root Mean Square Error (RMSE)

$$RMSE = \sqrt{\frac{1}{N}\sum_{r_{ij}}\left(r_{ij} - p_{ij}\right)^2}$$

N is number of available ratings r_{ij}

p_{ij} is the predicted ratings

A simple standard metrics to evaluate globally the quality of a recommend system is RSME. It compares the known ratings to the predicted ratings, using l2-norm.

RMSE

RMSE corresponds to the optimization objective used in matrix factorization

RMSE can be evaluated

- Globally, averaging over all ratings
- per-user/per-item with averaging over all users/items

RMSE can be evaluated using different strategies, either globally over all items, or first by computing the RMSE for items/users separately and then averaging the resulting values.

Evaluating Ranking Quality

Frequently only the most relevant items are recommended: evaluate the quality of the top-k recommendations

Various methods

- Non-discounted cumulative gain (NDCG)
- Mean reciprocal rank
- Hit rate

However, RMSE is not an adequate metric, if one is only interested into the top-k recommendations, as they are usually provided in practice. Therefore, alternative ranking metrics that consider the quality of the top-k recommended items are used.

Among those, we introduce in more detail the NDCG.

Evaluating Ranking Quality

Evaluating the top-k results returned for a user

- Let $r(i)$ be the score given for item i

Cumulative gain (CG): $\mathrm{CG} = \sum_{i=1}^{k} r(i)$

- Does not consider position of recommended items

Discounted cumulative gain (DCG):

$$\mathrm{DCG} = \sum_{i=1}^{k} \frac{r(i)}{\log_2 i + 1}$$

NDCG builds on the concepts of cumulative gain (CG) and discounted cumulative gain (DCG).

With CG the ratings of the top-k rated items are added up. The higher this value, the higher the quality of the predicted ratings.

CG does not consider the position of the rating, whereas in practice the earlier positions are more important. Therefore, with DCG a weighting function for the positions is included, which gives higher weight to the earlier positions. The method has its background in finance, where for investments gains in earlier years are higher weighted than gains further into the future.

Normalized DCG (NDCG)

Values of DCG are not normalized

- depend on rating distribution and choice of k

Compute the DCG for the top-k items sorted in optimal order, i.e., by decreasing scores:

$$iDCG(k)$$

Then normalize the DCG: $NDCG = \frac{DCG}{iDCG(k)}$

Finally, the metrics provided by DCG is not normalized and can thus not be compared for different rating schemes and choices of k. To calibrate the values, the DCG for the optimally sorted items is considered as baseline. By normalizing the DCS using this baseline, one obtains the NDCG, as a standardized measure for ranking quality in the interval [0,1].

Example

Assume the following score distribution:

(3,3,3,2,2,1)

Two methods A, B recommending top-3 items

A recommends (2,3,3)

B recommends (3,3,2)

Then $CG(A) = CG(B)$ but

$$DCG(A) = \frac{2}{\log_2 2} + \frac{3}{\log_2 3} + \frac{3}{\log_2 4}$$

$$< \frac{3}{\log_2 2} + \frac{3}{\log_2 3} + \frac{2}{\log_2 4} = DCG(B)$$

The example illustrates the different steps in the calculation of NDCG.

Example

Then

$$iDCG(3) = \frac{3}{\log_2 2} + \frac{3}{\log_2 3} + \frac{3}{\log_2 4} = 6.39$$

Therefore, normalized values

$$NDCG(A) = \frac{5.39}{6.39} = 0.84$$

$$NDCG(B) = \frac{5.89}{6.39} = 0.92$$

In this step the DCG values are normalized.

Question

Assume in top-1 retrieval recommendation 1 is (2, 3, 1) and recommendation 2 is (2, 1, 3)

1. RMSE(rec 1) > RMSE(rec 2) and DCG(rec 1) > DCG(rec 2)
2. RMSE(rec 1) = RMSE(rec 2) and DCG(rec 1) > DCG(rec 2)
3. RMSE(rec 1) > RMSE(rec 2) and DCG(rec 1) = DCG(rec 2)
4. RMSE(rec 1) = RMSE(rec 2) and DCG(rec 1) = DCG(rec 2)

Outlook: Bayesian Personalized Ranking

Ranking objective functions

- In practice recommender systems do not predict scores, but rankings

Instead of optimizing RMSE (as with matrix factorization) we can optimize the ranking for a user i by

- Sampling: a positive item j (which the user chose) and a negative item $\hat{j}$ (which the user never interacted with)
- Learning by maximizing the difference between their predicted scores: $\log \sigma(r_{ij} - r_{i\hat{j}})$
- In this way, the score is meaningless, only the order matters.

Finally, the absolute rating values are often not essential for recommendation. What counts are the items being proposed. Therefore, more recent approaches abandon RMSE as an optimization objective, and replace it with a metrics that essentially distinguishes only whether items are being selected by users or not. Bayesian Personalized Ranking is a representative of this class of approaches. We will encounter this method in more detail later in another context.

Summary

Recommender Systems evolved, similarly as IR, from basic models, to models based on optimizing an objective function

- Scaling to large numbers of users is a critical issue
- Cold-start problem is not solved by the more advanced models

The evolution of models for recommender systems was similar to that from models for information retrieval. Whereas IR models have to deal with scaling to large number of documents, RS have to take into account scaling to large numbers of users.

One problem that remains open, also with the advanced models, is the cold start problem. It is fundamentally not possible to make predictions on future ratings from historical ratings, if no historical ratings are available. Only using additional contextual information this problem can be solved.

References

- Chapter 9 in mmds.org
- Recommender Systems Handbook, Springer 2015
- Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 42.8 (2009): 30-37.
- Ning, Xia, and George Karypis. "Slim: Sparse linear methods for top-n recommender systems." *2011 IEEE 11th International Conference on Data Mining*. IEEE, 2011.
- Rendle, Steffen, et al. "BPR: Bayesian personalized ranking from implicit feedback." Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence. 2009.