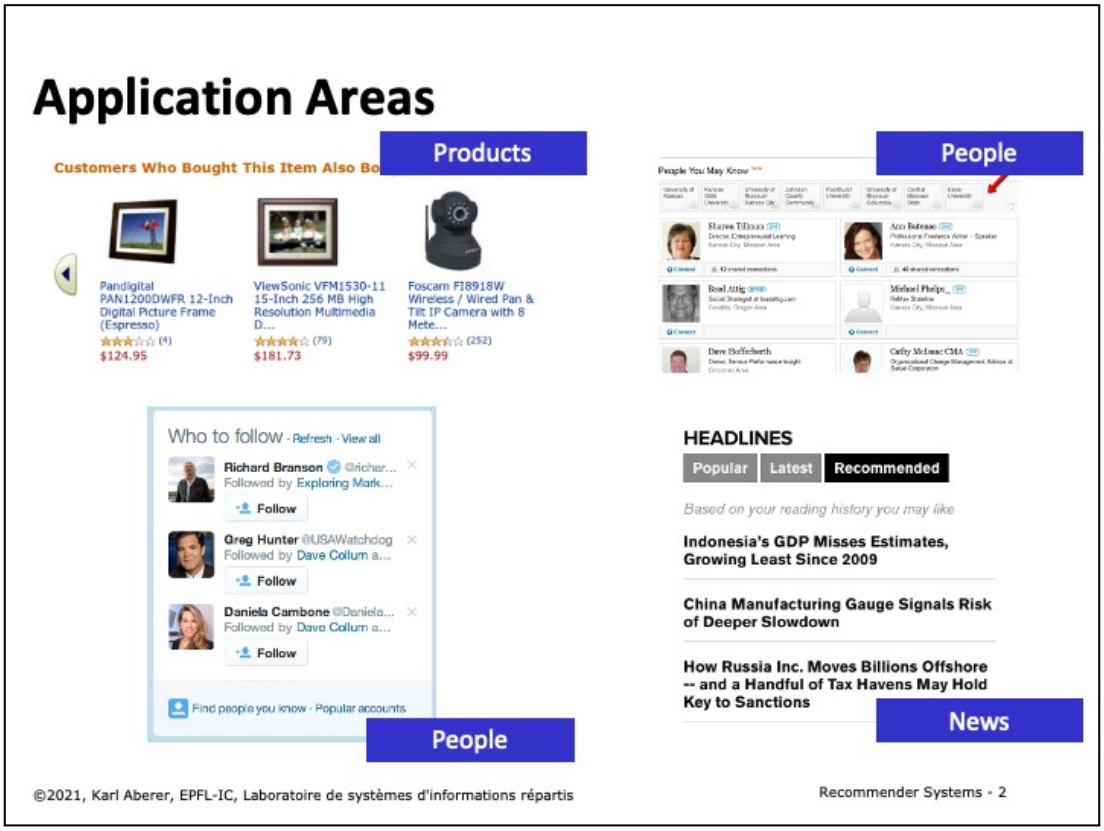
RECOMMENDER SYSTEMS



Recommender systems are widely used by many websites and 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

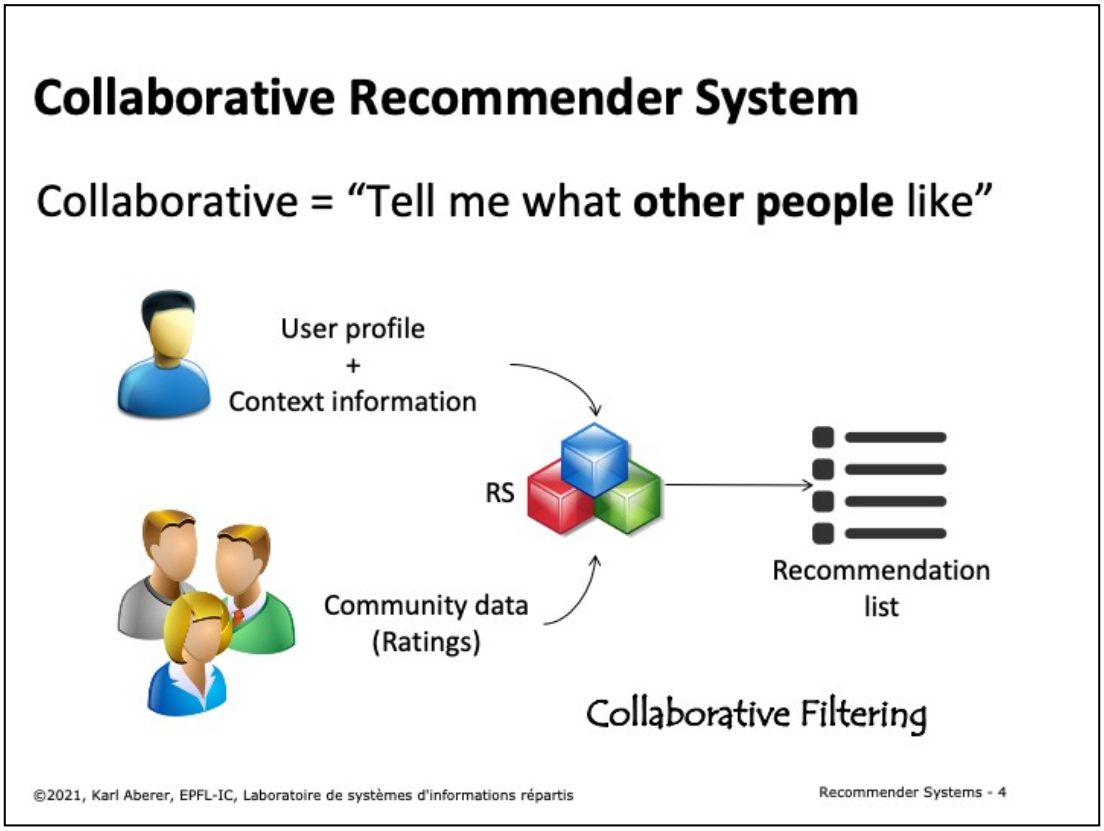
Definition: Recommender System

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 decreased *relevance*

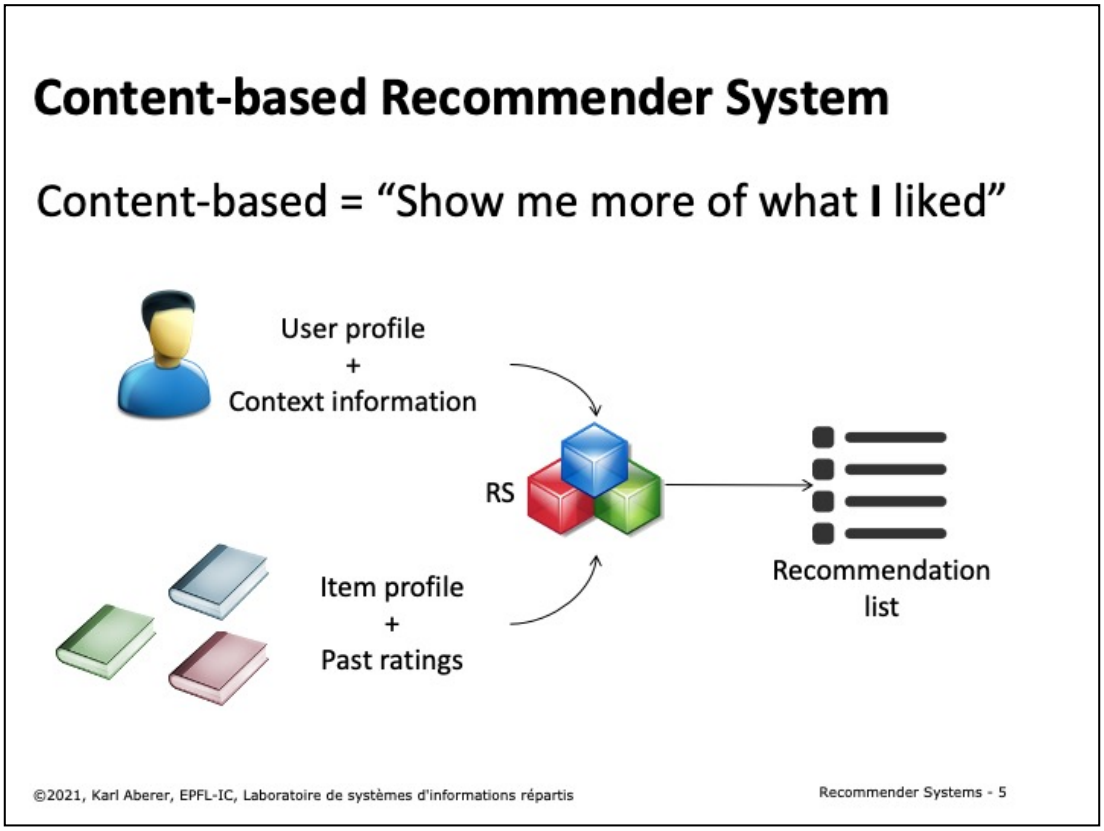
Learning a ranking function

A recommender systems identifies for a given user a set of items that are relevant to her. This can result in a ranking of the items (modelling a relevance function) or a classification.

The user model typically includes data such as ratings, preferences, demographics and attributes that characterize the user The items can also be enriched with attributes about their content and characteristics, although it is not always necessary.



In the collaborative paradigm, the recommender system uses the user profile (and possibly other information about the specific context in which the user needs a recommendation) and the data provided by other users to return a recommendation list with the most relevant items and their scores, from the highest (i.e., most relevant) to the lowest (i.e., least relevant). It is called collaborative because all the user participates to the recommendation by providing ratings to the items that they viewed/purchased etc.



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, from the highest (i.e., most relevant) to the lowest (i.e., least relevant). It is called content-based because the items are not only objects, but they are defined by a set of attributes that summarize the content of the item.

There are also other approaches, such as hybrid ones, which combine collaborative and content-based, or knowledge-based ones, which require domain knowledge engineering to model how certain item features meet the user needs.

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
- 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.

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

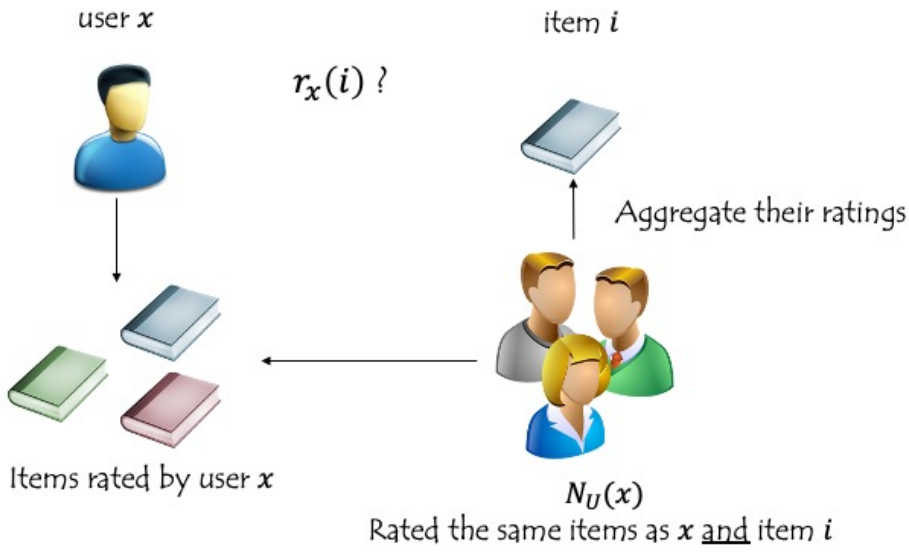
1. Finding a set of users $N_U(x)$ who rated the same items as x (= neighbours of x) in the past and who have rated i
2. Aggregate the ratings of i provided by $N_U(x)$

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

The basic technique for user-based collaborative filtering needs to define

1 A metric to compute similarity between users
2 The number of neighbours considered for $N_U(x)$
3 A way to aggregate the ratings of I provided by $N_U(x)$

User-based Collaborative Filtering illustrated



Are all users in $N_U(x)$ the same or are some more relevant than others?
➔ similarity between users

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

A widely used similarity measure is the Pearson correlation coefficient, which returns a value between -1 and 1. The cosine similarity considers the users x and y as two N-dimensional vectors. Computing similarity follows exactly the same principles as for document similarity in vector space retrieval. If the angle between the two vectors is 0, thus the users are identical, and the cosine similarity returns 1. The greater the angle, the less similar the two users are. Assuming that all ratings are positive, the maximum angle is π/2, thus the minimum similarity is cos(π/2)=0.

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, if the Pearson correlation coefficient is used, or User 2, if the cosine similarity is used. Also User 4 is not that different from U using the cosine similarity, although he seems to have quite opposite tastes.

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

Aggregate the Ratings

A common aggregation function is

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

$N_U(x)$: neighbours of user x

a : item not rated by x

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

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

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

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

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, that is, the recommendation for a new user that did not rate anything yet or for an item that was never rated. Also, user-based collaborative filtering is problematic when there are millions of users and/or items (Amazon, Netflix). For example, for each user the most similar users have to be found (nearest neighbours) from a large set of users. With large numbers of users, 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 $r_x(i)$

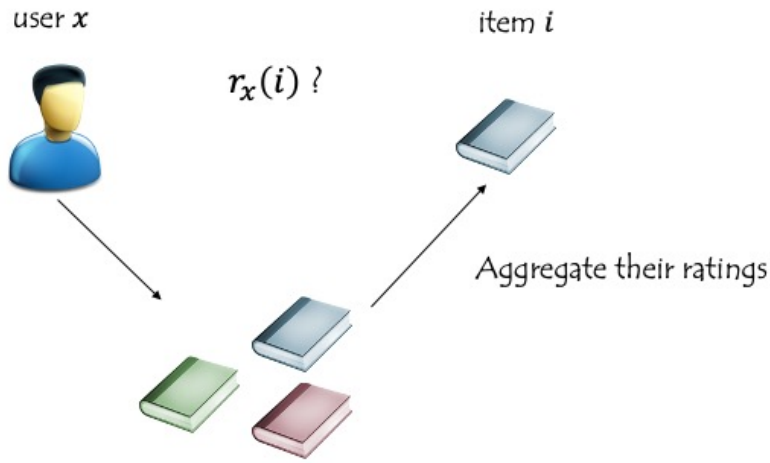
The basic technique for item-based collaborative filtering needs to define

1 A metric to compute similarity between items

2 The number of neighbours considered in N_I

3 A way to aggregate the ratings of the items in N_I

Item-based Collaborative Filtering illustrated



Items similar to the item i and rated by user x

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 explained before, 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

$$r_x(a) = \frac{\sum_{b \in N_I(a)} sim(a,b) r_x(b)}{\sum_{b \in N_I(a)} |sim(a,b)|}$$

$N_I(a)$: neighbours of item a

b : item rated by x

The aggregation function computes the prediction of an item a for a user x by computing the sum of the ratings given by the user on the items similar to a. Each rating is weighted by the corresponding similarity sim(a,b) between items a and b.

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

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

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 $N_I(a)$ 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, 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 $N_I(a)$ 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

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 need only the ratings of the user, and no information about the item is needed. However, the content of the item might provide useful information for the recommendation. Content-based recommendation takes into consideration only the items that have been rated by the user and the content of all existing items, significantly reducing the scalability problems since a community of users is not necessary any more.

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 a and a textual description t (e.g. movie synopsis, book review), compute the tf-idf weights

$$w(t,a) = tf(t,a) \cdot idf(t) = \frac{freq(t,a)}{\max_{s \in T} freq(s,a)} \cdot \log\left(\frac{N}{n(t)}\right)$$

N : number of items to recommend

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

The first step for content-based recommendation is the characterisation of the items in terms of TF-IDF. Each item can be therefore described by the set of terms that appear in their description, with their associated weight (the TF-IDF measure)

To compute the TF-IDF measure, all the steps typical of information retrieval (removing stopwords, stemming, only top M terms etc.) are applied.

Similarity between Items

Cosine similarity

$$sim(a,b) = \cos(\vartheta) = \frac{\sum_{t=1}^{T} w(t,a) \cdot w(t,b)}{\sqrt{\sum_{t=1}^{T} w(t,a)^2} \sqrt{\sum_{t=1}^{T} w(t,b)^2}}$$

a, b: items

T : number of terms t that appear in all items

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

The cosine similarity in content-based recommendation considers the items a and b as two T-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 $N_I(a)$ in D of a new item a

Use the ratings of the nearest neighbours to predict the rating of a with the aggregation function seen before

$$r_x(a) = \frac{\sum_{b \in N_I(a)} sim(a,b) r_x(b)}{\sum_{b \in N_I(a)} |sim(a,b)|}$$

Once a way to compute the similarity between items has been established, using the vector space model (TF-IDF + cosine similarity), the simplest approach to estimate the rating of item a not yet seen/rated by user x is to find the nearest neighbours of a in the subset of items that have been already rated by the user x and aggregate their ratings. Note that this approach to predict the rating is analogous to the use of the kNN nearest neighbor 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 limited 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.

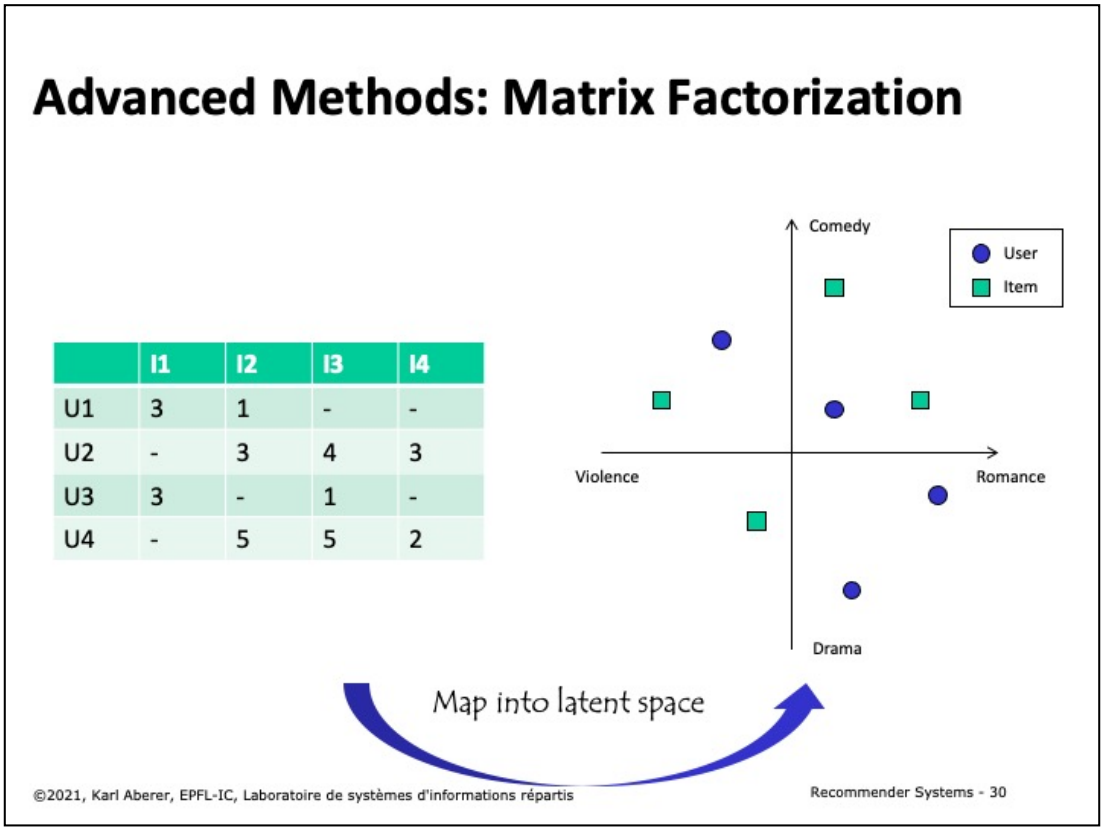
On the other hand, it is in general more scalable, because it does not rely on a community of users to provide recommendations. Furthermore, the TF-IDF measure can be computed once a new item enters the system, and then update the pair-wise similarities with all the existing items.

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



	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 model, where each user and item is described as a vector in a k-dimensional space. It can be understood as a combination of user-based and item-based collaborative filtering.

For the items, the factors describe evident as well as non interpretable characteristics. For the users, each factor measures how much the user likes items that score high on the corresponding factor. The estimate of the rating of item a that user u would given can be computed as the dot product between the two low-dimensional matrices: $q_i^T \cdot p_u$ where q_i is the item vectors and p_u is the user vectors. The major challenge of course is computing the mapping of each item and user to their corresponding vectors in q_i and p_u. After that, the rating can be estimated using the dot product.

This approach is analogous to singular value decomposition for document retrieval. However, a direct application of SVD is not possible because the user-item matrix is not complete (missing ratings).

Derive Latent Factors

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

- Users U and Items D

Goal: Decompose R (approximatively)

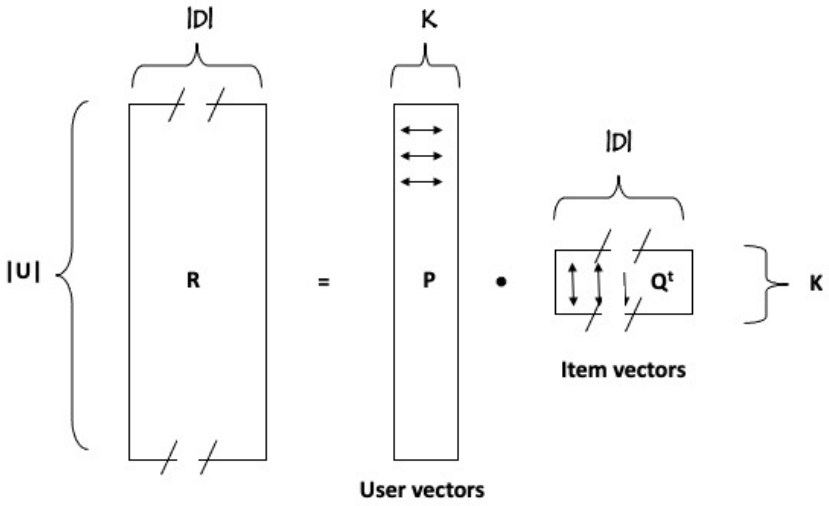
$$R \approx P\times Q^t = \hat{R}$$

K latent factors (low dimension)

- P is a $|U|\times K$ matrix: user features
- Q is a $|D|\times K$ matrix: item features

Here we formulate the factorization problem.

Illustration of Matrix Factorization



Computing Matrix Factorization

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

- Matrix decomposition (as used for LSI) not applicable

Formulate an optimization problem

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

Apply SGD

Approximation Error

For a given rating r_{ij}

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

Minimizing error: compute gradient

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

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

Thus, to learn the factor vectors q_i and p_u the system minimises the regularised square error on the set of known ratings $(u,i) \in M$. The minimization problem can be solved using stochastic gradient descent. To that end we first compute the gradients.

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

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

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

In order to avoid over-fitting, usually a regularization term is added and the update rule is modified correspondingly.

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 scales with the number of users, which in real world scenarios can become very large (e.g. customers of an ecommerce or streaming service).

SLIM, Sparse Linear Methods

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

- Items with similar ratings have similar representation

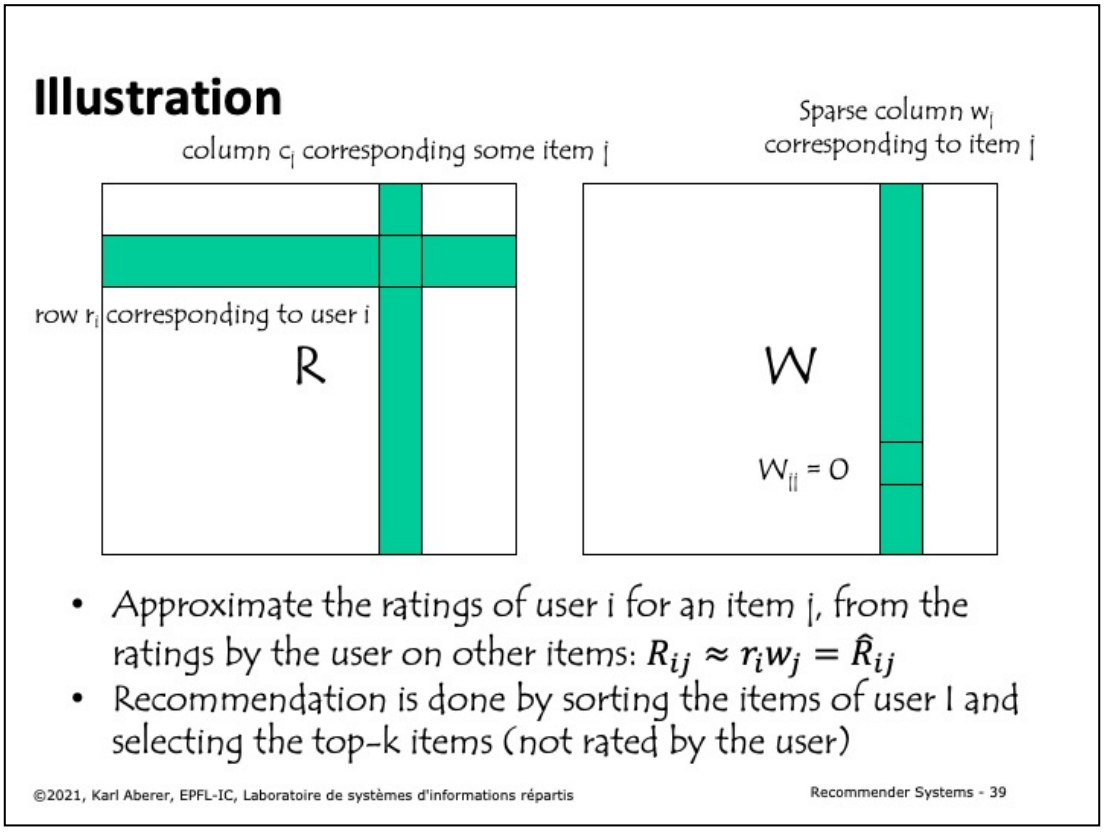
Approximate Rating Matrix R with dimension $|U|\times|D|$ using a **sparse** weight aggregation matrix W

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

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

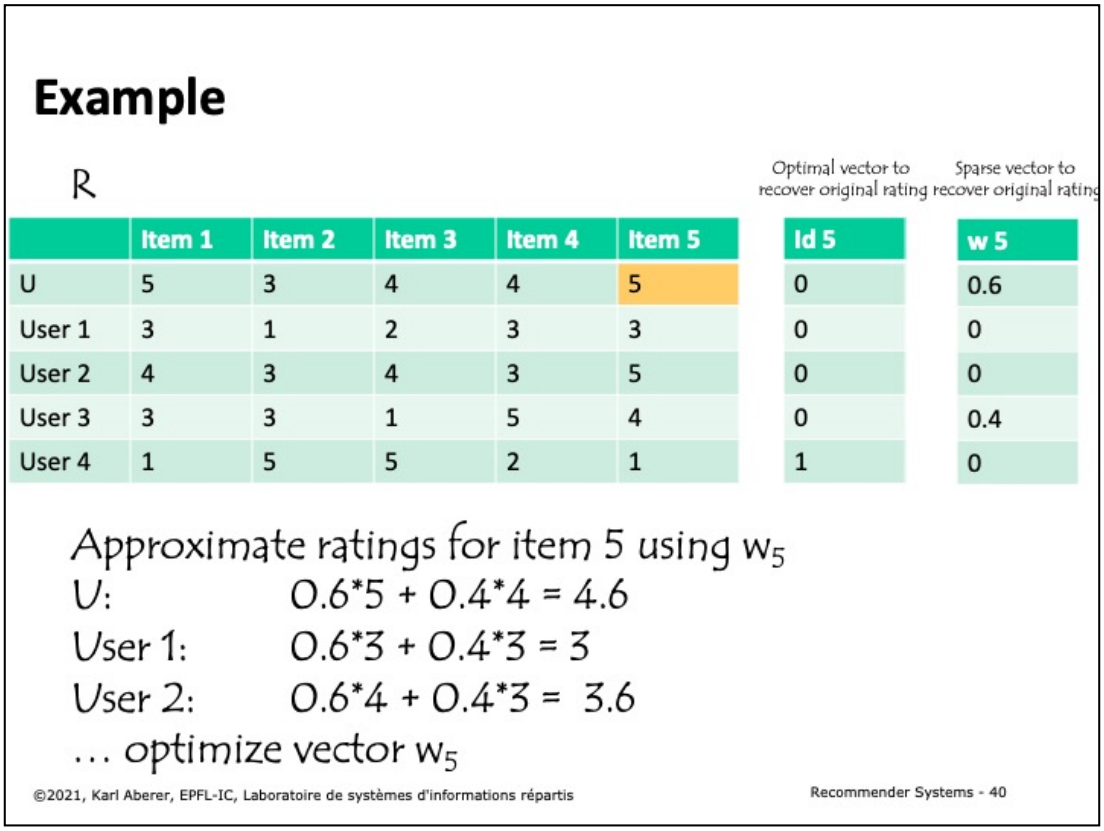
In order to overcome the issue of scaling with the user population, alternative models have been considered. SLIM is a representative of those.

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 idea 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.



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.

In typical experiments, only about 1% of the values of W would be non-zero, which explains why the method can make recommendations faster, since the product $r_i w_j$ can be computed fast.



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 (e.g. item 5) a combination of other items that allow to reconstruct the ratings of item 5 for each user (e.g. items 1 and 4 to reconstruct item 5). This vector cannot contain a 1 on the diagonal, otherwise it would be trivial to reconstruct the rating of item 5.

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)

In order to enforce sparsity of the matrix in the loss function the 1-norm of W is minimized. In addition, in order reduce the size of weights the 2-norm is minimized. These are standard techniques from machine learning.

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 $W \geq 0$

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)

Since the columns of W are independent the problem can be solved in parallel for each item. For this standard coordinate descent methods can be applied (i.e. 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
- Training can be performed with a subset of users
- Users need not to be individually modelled, only their rating history needs to be stored

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

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

We will among those introduce in more detail the NDCG.

Evaluating Ranking Quality

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

Evaluating the top-k results returned for a user

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 cumulative gain (CG) and discounted cumulative gain (DCG).

With CG the scores of the top-k rated items are added up. In order to consider also the position, those scores are weighted with a position factor, such that with the same scores the ranking wins, that puts the higher scored items into higher positions.

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 with changing rating grades and choices of k. to compensate for that, 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

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

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

Outlook: Bayesian Personalized Ranking

Ranking objective functions

- In practice recommender systems to 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 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.

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

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.