Recommender Systems

Zheng Maria Yu

Politecnico di Milano, 2022/23

Contents

1 Introduction

A **recommender system** is a decision-support system which filters and analyses input data. It facilitates the users by providing them with hints and suggestions about items that can meet their interest.

1.1 Sources of information for recommender algorithms

- A list of available items with their descriptions, and eventually a set of attributes;
- User attributes such as demographics;
- Interactions between users and items;
- Context (contextual attributes of the interactions).

1.2 Taxonomy of recommender systems

- **Non-personalized**: all users receive the same recommendations.
- **Personalized**: different users receive different recommendations.
  - **Content-Based Filtering** (CBF): its goal is to recommend items similar to the ones a user liked in the past. For content-based recommendations, it is important to have a list of good quality attributes for each product.
  - **Collaborative Filtering** (CF): it relies on the opinions of a community of users, no item attributes are required.
    - **User-based**: it focuses on user-user similarities, searching for users sharing the same opinions on a number of items.
    - **Item-based**: its basic idea is to calculate the similarity between each pair of items, according to how many users have the same opinions on them.
    - **Matrix Factorization**: it is an example of dimensionality reduction algorithms.
  - **Context-Aware** (CARS): it extends collaborative filtering algorithms by using the context to improve the quality of recommendations.
  - **Hybrids Side Information**: also called "collaborative filtering with side information", it uses simultaneously multiple and heterogeneous sources of information to enrich users, items, and interactions with side information.

Other techniques, such as Factorization Machines or Deep Learning techniques, could be used as collaborative filtering algorithms, and they perform better with contextual or other sources of side information.

1.3 Significant matrices

- **Item Content Matrix** (ICM): it describes the list of items with the attributes, with rows representing items and columns representing attributes. Each number in the ICM indicates how much important an attribute is in characterizing an item. In the simplest form, the value in is ICM are 1 if the item contains that specific attribute, or 0 otherwise. Generally speaking, any positive value can be assumed.
- **User-Rating Matrix** (URM): $R = [r_{ui}]$, it describes the past interactions between users and items. Rows represent users, columns represent items and numbers are implicit or explicit ratings. If there is no information about the opinion of the user on a specific item, the corresponding value will be set to 0.
  - **Explicit rating**: the user was asked to rate an item on a certain scale. 0 indicates that there is no information on that item for that user.
    - Large rating scale: precise opinion, more user effort, fewer ratings;
    - Small rating scale: more ratings, lower rating quality;
    - Even rating scale: no neutral element, fewer ratings;
    - Odd rating scale: more ratings, but users could prefer to give neutral ratings which are useless.

    On the other hand, users tend to publish their rating only if they had a positive experience. This could create a bias affecting the rating distribution.
  - **Implicit rating**: $r_{ui} \in \{0, 1\}$, the interaction is marked as 1 if we think that the user was interested in the item, 0 otherwise. Implicit ratings are obtained according to specific criteria based on users' behaviour, without asking for an opinion explicitly.

  The URM is a huge matrix but it is sparse (typical density $< 0.01\%$), and it can be managed in an efficient way.
  The goal of any recommender algorithm is to predict missing values in the URM (matrix completion task).

2 Non-Personalized Recommenders

2.1 Top Popular

Taking the URM, the most popular items are those which have been rated the greatest number of times (without considering the opinion of the users).

2.2 Best Rated

Taking the URM, the best rated items can be found by computing the **average rating per item**:

$$b_i = \frac{\sum_u r_{ui}}{N_i}, \tag{1}$$

with r_{ui} the rating given by user u to item i (non zero ratings) and N_i the number of users who rated item i.

To take into account the bias due to the number of users who have rated the item, a shrink term C can be introduced. It is a constant value, chosen depending on the properties of the URM. So we have **shrinked average rating** for item i:

$$b_i = \frac{\sum_u r_{ui}}{N_i + C}. \tag{2}$$

2.3 Global Effects

This technique can be used to make simple but effective recommendations when we have explicit ratings, or to normalize the URM before using more advanced algorithms.
To make predictions, the biases of items (some items have larger ratings than other ones) and of users (some users give more generous ratings than other ones) are computed and removed during the procedure.

1. Compute the overall average ratings for all items and users:

$$\mu = \frac{\sum_i \sum_u r_{ui}}{N}, \tag{3}$$

   with r_{ui} the explicit rating given by user u to item i, and N the total number of non zero ratings. This term is called the **global bias**.

2. Compute the normalized rating for each user u and item i, only on non-zero ratings, by removing the global bias:

$$r'_{ui} = r_{ui} - \mu. \tag{4}$$

3. Calculate the **item bias**:

$$b_i = \frac{\sum_u r'_{ui}}{N_i + C}. \tag{5}$$

   N_i is the number of users who have rated item i, and C is the shrink term.

4. Remove the item bias (only on non-zero ratings):

$$r''_{ui} = r'_{ui} - b_i. \tag{6}$$

5. Introduce the **user bias**:

$$b_u = \frac{\sum_i r''_{ui}}{N_u + C}, \tag{7}$$

for the fact that users could be more or less generous in their ratings. N_u is the number of items i rated by user u.

6. The Global Effects final formula is

$$\tilde{r}_{ui} = \mu + b_i + b_u. \tag{8}$$

It allows to create a new unbiased URM.

It is considered a non-personalized method even if each user receives a different recommendation, because of the ease of calculation. Moreover, considering the top-n items, the selection is the same for all the users.

3 Quality of Recommender Systems

The quality of a recommender system basically depends on the dataset, the algorithm, and the user interface. To test the effectiveness of the recommender system, the online evaluation allows to measure the quality of all the three aspects, while the offline evaluation only tests the quality of the dataset and the algorithm.

A recommender algorithm is based on the functions f(URM), and g(model, user profile). The output of f is the model, i.e. a representation of the user preferences. g estimates ratings from the model and the user profile (past explicit/implicit ratings of the user). At the end, the recommendation is compared with the true opinion of the user.

3.1 Quality indicators

- Relevance: ability of recommending items that very likely the user will appreciate. It is our main focus and it is the easiest to measure.
- Coverage: ability of recommending most of the items of a possibly large set of items. It measures the percentage of items that a recommender system is able to recommend. It is important from the point of view of providers.
- Novelty: ability of recommending items unknown to the user.
- Diversity: ability to diversify the recommended items.
- Consistency: it stands for the stability of the recommendations. If they change too often between sessions, the user might be disoriented.
- Confidence: ability of measuring how much the system is sure about a recommendation. Not all recommender algorithms are able to provide confidence estimates.
- Serendipity: ability of surprising the user, who can discover something unexpected.

3.2 Online evaluation

Online evaluation consists in asking the user satisfaction, and measuring the quality perceived by the user. Some online evaluation techniques are:

- **Direct user feedback**: it asks some real users (the size of the sample should be meaningful) to define their level of satisfaction about the recommender system. However, the opinion expressed by the user could not be reliable, and the cost should be also considered.
- **A/B testing**: the behaviour of users who receive recommendations (set A) is compared with the behaviour of the users who do not receive them (set B). It is a powerful and effective evaluation method, but it is difficult to set up and its results may be difficult to interpret.
- **Controlled experiments**: a mockup application is offered to a group of potential users, which are asked to give their opinion. Its problems are that the mockup application is not the real one, and the users are not real nor motivated. Their opinions can be unreliable.
- **Crowdsourcing**: people are asked to express their opinion about the mockup of an application, under some kind of compensation. There could be a large "crowd" of "volunteers", but very strong statistical tests are required to understand the reliability of their answers.

3.3 Offline evaluation

To analyze offline evaluation techniques, some important aspects should be considered.

3.3.1 Task of the recommender system

- **Rating prediction**: the goal is to approach the true value of the rating.
- **Top-N recommendation**: the goal is to find N items relevant for the user. Typically items are ranked from the most relevant to the least one, and the first N items of this list are selected.

3.3.2 Dataset

Speaking about recommender systems, the dataset is always represented by the URM. Its (small) known part containing all non-zero ratings is called "**ground truth**", which is composed by a relevant part (all the positive opinions given by the users) and a non-relevant part (all the negative opinions). On the other hand, the unknown ratings consists in the non-rated interactions.

There are different approaches to deal with the dataset:

- Data interpolation: use an expression that fits perfectly the dataset. Such a model has no generalization ability, and it is unable to predict on new data. In this case we are exposed to the phenomenon of overfitting, which generally occurs when dealing with a high number of parameters.
- Data extrapolation: try to generalise the function that approximates the dataset. In this case there is overfitting.

3.3.3 Dataset partitioning

- **Hold Out of Ratings**: some interactions are extracted from the dataset and kept away in a set Z (test set). It can be built by randomly choosing a percentage of the ratings, or by selecting only one interaction per user (leave one out). The remaining ratings - the training set X - are used as input for f. The user profile set Y ($Y \subseteq X$) and the model are fed to g. At the end, the estimated results are compared with the ones present in Z.
- **Hold Out of Users**: the training set is built by selecting some users and all their interactions, and some interactions among the left users are put in the test set Z. These users' other interactions constitute the user profile set Y (now Y is not a subset of X).

3.3.4 Quality metrics

Every system needs to be evaluated with the appropriate metric, and it does not exist a perfect rule.

- **Error metrics**: their purpose is to estimate the difference between ratings assigned by algorithms and the one assigned by users: $e_{ui} = |r_{ui} - \hat{r}_{ui}|$, with r_{ui} the true rating in the test set and $\hat{r}_{ui}$ the estimated rating.

  Some methods are used to evaluate the goodness of the recommender system.

  - **Mean Absolute Error**:

    $$MAE = \frac{\sum_{u,i \in T} |r_{ui} - \hat{r}_{ui}|}{N_T}. \tag{9}$$

    T is the test set, and N_T the number of interactions in the test set (non-zero ratings). MAE is the overall average of the absolute value of errors.

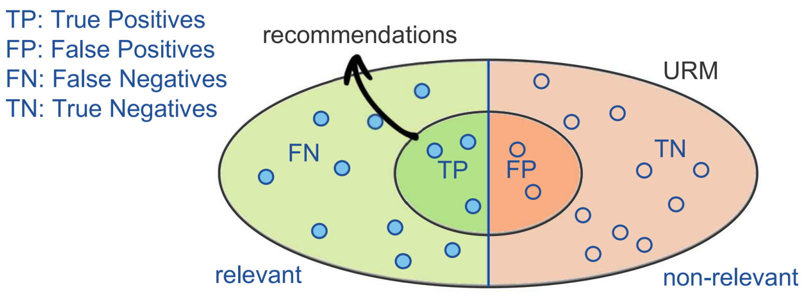
  - **Mean Squared Error**:

    $$MSE = \frac{\sum_{u,i \in T} (r_{ui} - \hat{r}_{ui})^2}{N_T}. \tag{10}$$

    Sometimes the root of the result is considered for normalization purpose. The MSE is differentiable, and it is easier to minimize through performing gradient descent.

The error can be measured if the recommendations overlap with the ground truth. When an error metric is used, it is computed on non-zero elements making the **Missing As Random** (MAR) assumption: the distribution of unknown ratings is identical to the distribution of the ground truth. This is not realistic.

- **Classification metrics**: the goal is to determine if a user likes an item or not. Taking an URM divided into a set of relevant items and a set of non-relevant ones, we highlight the K suggested items and we define the quantities:
  - True positives (TP) are all the relevant items recommended to the user. The number of true positives is also referred to as the number of hits.
  - False positives (FP) are all the non-relevant items recommended. The number of non-relevant recommended items is often called miss.
  - False negatives (FN) are all the relevant items not recommended.
  - True negatives (FN) are all the non-relevant items not recommended.



The classification metrics depend on the number K, and some examples are:

  - **Recall** at K:

$$\text{Recall(K)} = \frac{\#\text{relevant recommended items}}{\#\text{tested relevant items}} = \frac{\text{TP}}{\text{FN+TP}}. \tag{11}$$

  Also known as **sensitivity** or **True Positive Rate**, it explains the percentage of positive instances correctly detected by the system in all the items relevant for the user.

  - **Precision** at K:

$$\text{Precision(K)} = \frac{\#\text{relevant recommended items}}{\#\text{all recommended items}} = \frac{\text{TP}}{\text{FP+TP}}. \tag{12}$$

  It values the number of hits in relation to all the K recommended items, and it explains the accuracy of the predictions.

- **Fallout** at K:

$$\text{Fallout(K)} = \frac{\text{\#non-relevant recommended items}}{\text{\#all non-relevant items}} = \frac{\text{FP}}{\text{FP+TN}}. \tag{13}$$

  It is also called **False Positive Rate** and it can be computed only if the negative ratings are explicitly defined. It is supposed to be as low as possible.

Increasing the value of K, generally both Recall and Fallout increase, meanwhile the precision decreases.

Since that the produced recommendations may be overlapped with the ground truth or not, we can say that they are relevant only if they refer to the ground truth, and all the others are considered as non relevant. This assumption for the classification metrics is called **Missing As Negative** (MAN).

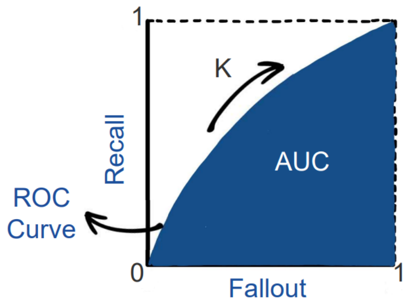
- **Ranking metrics**: their purpose is to measure how much a user likes an item with respect to the other ones. An ordered list of recommendations is presented to the users, and the most relevant items are put at the top.

  - **AUC metric**: the **Receiver operating characteristic** (ROC) curve is plotted with Fallout on the x-axis and Recall on y.

    In the case of explicit ratings, the **Area Under the Curve** (AUC) can be computed:

$$AUC = \frac{\sum_k \text{Recall}(k) * \Delta\text{Fallout}}{\text{\#items}}. \tag{14}$$

    AUC has value 1 for a perfect recommender system, and 0.5 for a random recommender. The goal is to get the ROC curve as near as possible to $(0, 1)$.

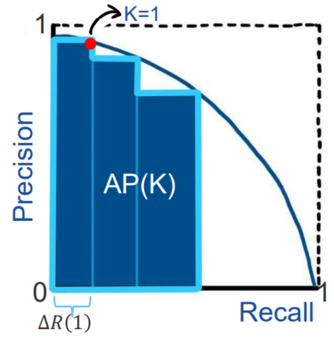


  - **Mean Average Precision** (MAP): in the Precision-Recall curve, the Recall values are plotted on the x-axis and the Precision on y. The more it approaches towards $(1, 1)$, the better it is.
    The **Average Precision** (AP) is the area under the Precision Recall curve, and it can be computed approximately by dividing it in rectangles for every value of

K:

$$AP(k) = \sum_k P(k) \cdot \Delta R(k) = AP(k-1) + \Delta R(k) \cdot P(k), \tag{15}$$

with $R(0) = 0$ for convention, and $\Delta R(k) = R(k) - R(k-1)$.



So we have

$$MAP(k) = \frac{\sum_u AP(k)_u}{N_u}, \tag{16}$$

where $AP(k)_u$ is the Average Precision at K computed for user u and N_u the number of users.

MAP acts like a corrected version of precision by giving more importance to the bits of information we have.

– **Average Reciprocal Hit-Rank** (ARHR):

$$ARHR = \frac{\sum_i \frac{1}{\text{rank}(i)}}{\#\text{tested relevant items}}, \tag{17}$$

with $i \in$ relevant recommendations and rank(i) representing the position of item i in the list of recommendations.

It is a modified version of recall, but it is not a ranking in strict sense since it does not compare the ranking provided by the user with the one provided by the system.

4 Content-Based Filtering

In content-based filtering, the main assumption is that a user that expressed a preference for an item will probably like similar items. The items are compared and recommended based on their attributes, which are represented in the ICM.

4.1 Cosine Similarity

This technique is often used to measure the similarity between items. Let us consider two items i and j as two binary vectors (with values 0 or 1) in the ICM. The number of common elements between them can be computed with the Dot Product:

$$\vec{i} \cdot \vec{j} = \sum_a i_a j_a = \# < i, j > . \tag{18}$$

The normalized dot product

$$s_{ij} = \frac{\vec{i} \cdot \vec{j}}{|\vec{i}|_2 \cdot |\vec{j}|_2} = cos(\theta) \tag{19}$$

leads to compute the similarity with the cosine of the two vectors of attributes. Graphically, it is the angle included between the two vectors.

To take into account the support of the vectors (number of non-zero elements in them, larger supports are statistically more significant), the shrink term can be introduced in the computation:

$$s_{ij} = \frac{\vec{i} \cdot \vec{j}}{|\vec{i}|_2 \cdot |\vec{j}|_2 + C}. \tag{20}$$

Its value represents an hyperparameter.

4.2 Similarity matrix

The **similarity matrix** is composed by the similarities computed across all pairs of items: the element (i, j) in the matrix describes how much item i is similar to j. It is symmetric if we consider the similarity computed with the dot product or the shrinked cosine.

With the similarity matrix, it is possible to estimate the rating of user u on item i by relying on the past ratings from the same user on other items j. The weighted average of the previous given ratings is

$$\tilde{r}_{ui} = \frac{\sum_j r_{uj} \cdot s_{ji}}{\sum_j s_{ji}}. \tag{21}$$

The normalization at the denominator is useful if we want to estimate the ratings as accurately as possible, but it may not be necessary when doing a Top-N recommendation.

At this point, CBF can be described by using a matrix notation. When we consider the whole user profile, for each user u the estimated rating $\tilde{r}_{ui} = \sum_j r_{uj} \cdot s_{ji}$ is equivalent to $\vec{\tilde{r}_u} = \vec{r_u} \cdot S$, since the summation of the past ratings can be seen as a vector, and the similarity as a matrix (the entire items similarity matrix).

Generalizing the formula for all the users, the **estimated rating matrix** is $\tilde{R} = R \cdot S$, where R is the URM and S the similarity matrix.

4.3 K-Nearest Neighbours

A similarity matrix comes with some issues:

- Very dense: since that only few cells are empty, the matrix is computational expensive in both terms of memory and time.
- Small similarity values: the fact that the similarity values are generally small and very similar to each other can cause the presence of noise in the data. It leads to lower quality recommendations.

The KNN technique is introduced as a solution to simplify the problem. It consists in keeping only the K most similar couples of items in the similarity matrix. In fact, the formula for the estimation of the ratings now keeps track of the **K nearest neighbours**:

$$\tilde{r}_{ui} = \frac{\sum_{j \in \mathrm{KNN}(i)} r_{uj} \cdot s_{ji}}{\sum_{j \in \mathrm{KNN}(i)} s_{ji}}. \tag{22}$$

The parameter K influences the quality of the recommendations. If the value of K is too small, there is no enough data to make reliable estimations. On the other hand, if the value is too big, the data will contain too much noise.

4.4 Improving the ICM

In Content-Based systems, one of the most important driver of quality is the quality of the ICM, which can be improved by introducing **Non-Binary attributes**. An item, in fact, can have or not specific attributes in a different extent, and in this way the non-binary weights can be regulated according to the importance of each attribute.

The ICM can be further improved with **attribute weights**, which is an example of feature engineering. Different types of attributes could be given different weights.

4.5 Term Frequency - Inverse Document Frequency

These techniques aim to automatically balance the weights of attributes in the ICM, according to their frequency of appearance in the items.

In order to obtain more balanced values for the weights of the attributes, the **TF-IDF** is given by the product of the Term Frequency and the Inverse Document Frequency.

- **Term Frequency**:
$$TF_{a,i} = \frac{N_{a,i}}{N_i}. \tag{23}$$
$N_{a,i}$ stands for the appearances of attribute a in item i, and N_i is the total number of attributes of item i. Since $N_{a,i}$ is often equal to 1, a lot of attributes can have very little term frequency values.

- **Inverse Document Frequency**:

$$IDF_a = \log \frac{N_{\text{ITEMS}}}{N_a}. \tag{24}$$

  N_{ITEMS} is the total number of items, and N_a represents the number of items in which the attribute a is present. This term aims at solving the problem of little values of Term Frequency for rare attributes.

5 Collaborative Filtering

Collaborative filtering methods rely on the opinions of a community of users, and they do not require any item attribute to work.

The main input to a Collaborative Filtering algorithm is the URM containing past interactions between users and items. The ratings can be both explicit or implicit, but in general simple algorithms are designed to work with only one of the two types.

5.1 User-based techniques

The basic idea is to search for users with a similar taste, and to recommend the items these users like most. In the case of implicit ratings, user similarity can be measured by counting the common liked items: the similarity between user u and user v is $s_{uv} = \# < u, v >= \sum_i r_{ui} \cdot r_{vi} = \vec{u} \cdot \vec{v}$.

To have a measure of similarity with a number between 0 and 1, the **cosine similarity** can be used:

$$s_{uv} = \frac{\sum_i r_{ui} \cdot r_{vi}}{\sqrt{\sum_i {r_{ui}}^2 \cdot \sum_i {r_{vi}}^2}} = \frac{\vec{r_u} \cdot \vec{r_v}}{|\vec{r_u}|_2 \cdot |\vec{r_v}|_2}. \tag{25}$$

In case of implicit ratings, it can be formulated with respect to the number of interactions:

$$\frac{\# < u, v >}{\sqrt{\# < u > \# < v >}}. \tag{26}$$

When the support is small, it is opportune to add a shrink term C (maybe between 1 and 10) in the denominator of the cosine similarity formula.

All the similarity values between each pair of users can be summed up in a similarity matrix, with the cells describing how much user u is similar to user v. The diagonal elements are normally considered zero.

When using the **KNN** approach, for each user only the K most similar users to him are kept, and the value of K to maximize the quality of the recommender system should be tuned. At this point, the estimated rating that user u will give to item i could be simply computed as the weighted average of the ratings provided by other users on item i, where the weights are the similarities between users:

$$\tilde{r}_{ui} = \frac{\sum_{v \in \text{KNN}(u)} r_{vi} \cdot s_{vu}}{\sum_{v \in \text{KNN}(u)} s_{vu}}. \tag{27}$$

The items with the highest estimated ratings will be recommended.

When dealing with explicit ratings instead, the user bias and the item bias should be taken into account. By normalizing the ratings before the multiplication of the cosine formula, the **Pearson correlation coefficient** is obtained:

$$s_{uv} = \frac{\sum_i (r_{ui} - \bar{r_u}) \cdot (r_{vi} - \bar{r_v})}{\sqrt{\sum_i (r_{ui} - \bar{r_u})^2 \cdot \sum_i (r_{vi} - \bar{r_v})^2 + C}}, \tag{28}$$

with $\bar{r_u} = \frac{\sum_i r_{ui}}{N_u + C}$ and $\bar{r_v} = \frac{\sum_i r_{vi}}{N_v + C}$ the user biases (user's average opinion).

The delta (how much the user will like an item more or less than his average) should be considered for the ranking estimate: the final formula of the estimated rating becomes

$$\tilde{r}_{ui} = \frac{\sum_{v \in \text{KNN}(u)} (r_{vi} - \bar{r_v}) \cdot s_{vu}}{\sum_{v \in \text{KNN}(u)} s_{vu}} + \bar{r_u}. \tag{29}$$

5.2 Item-based techniques

In item-based approaches, the similarity between each pair of items is calculated according to how many users have rated them both.

Assuming to have an implicit URM, the similarity between item i and item j is measured as $s_{ij} = \# < i, j >= \sum_u r_{ui} \cdot r_{uj} = \vec{i} \cdot \vec{j}$ (number of common interactions).

The cosine similarity can be defined:

$$s_{ij} = \frac{\sum_u r_{ui} \cdot r_{uj}}{\sqrt{\sum_u {r_{ui}}^2 \cdot \sum_u {r_{uj}}^2}} = \frac{\vec{r_i} \cdot \vec{r_j}}{|\vec{r_i}|_2 \cdot |\vec{r_j}|_2}, \tag{30}$$

($= \frac{\#<i,j>}{\sqrt{\#<i>\#<j>}}$ in case of implicit ratings). Eventually a shrink term C can be introduced in the denominator.

Using the KNN approach, only the K-most similar items to a certain item are kept, and the items with the highest estimated rating are recommended:

$$\tilde{r}_{ui} = \frac{\sum_{j \in \text{KNN}(i)} r_{uj} \cdot s_{ji}}{\sum_{j \in \text{KNN}(i)} s_{ji}}. \tag{31}$$

For explicit ratings, the user bias term $\bar{r_u} = \frac{\sum_i r_{ui}}{N_u + C}$ should be considered. The **Adjusted Cosine** formula is

$$s_{ij} = \frac{\sum_u (r_{ui} - \bar{r_u}) \cdot (r_{uj} - \bar{r_u})}{\sqrt{\sum_u (r_{ui} - \bar{r_u})^2 \cdot \sum_u (r_{uj} - \bar{r_u})^2 + C}}. \tag{32}$$

5.3 Model-based vs Memory-based

Both model-based and memory-based methods are commonly used in collaborative filtering.

- **Memory-based techniques** were the earliest collaborative filtering algorithms. They are easier to implement: the values of the URM are directly used in the prediction, and they predict the ratings on the basis of user neighbourhoods.
- **Model-based techniques** do not rely on the whole dataset to compute recommendations. Instead, they extract information from the dataset to build a model.

5.3.1 Procedure

These techniques need two steps for the prediction.

1. Build the model: the model is equal to a function f taking as input a matrix (the URM in case of collaborative filtering, the ICM in case of content-based filtering).
2. Estimate ratings: the ratings are estimated using a function g(model, user profile), which takes as input the defined model and the user profile. The technique is memory-based if the user profile belongs to the URM, otherwise it is model-based.

In case of memory-based techniques, recommendations can be made only to users already present in the model, since each time they require to recompute the similarity between the potential new user and all the other users.

In case of model-based techniques, there is no need to update the URM for new users and to recompute the similarity matrix, if the URM is big enough. So it is possible to make recommendations to users not present in the model.

5.3.2 Applications in CF

In Collaborative Filtering, the model is a function f of the URM, while the estimated ratings are a function g of the model and the user profile.

In item-based CF, the estimated rating $\tilde{r}_{ui}$ is defined as in Equation 31. The similarity matrix S_{II} between items is used as the model for the function g, while the profile of user for which we are making recommendations, $\vec{r}_u$, is used as the user profile.

$$S_{II} = f(URM), \quad \text{estimated ratings} = g(S_{II}, \vec{r}_u). \tag{33}$$

Thus, the item-based collaborative filtering technique is model-based. In fact, adding a new user does not change the model significantly.

On the other hand, talking about user-based CF, the estimated rating $\tilde{r}_{ui}$ is defined as in Equation 27. In this case, the function g takes as inputs the similarity matrix S_{UU} and the user profile $\vec{r}_u$.

$$S_{UU} = f(URM), \quad \text{estimated ratings} = g(S_{UU}, \vec{r}_u). \tag{34}$$

It is a memory-based technique, since adding a new user implies the expansion and the recomputation of the similarity matrix.

5.4 Association Rules

A quite common technique used in data mining to compute the similarity between items consists in **Association Rules**. It is a probabilistic approach, and it is based on the idea of exploring the data that we have, in order to estimate the probability that something will happen if something else happened in the past.

The ideal case aims to compute $P(i|j_1, j_2, j_3...)$ which is the conditional probability that a user will like item i, by knowing the history of all the ratings the user gave to the items j_1, j_2, j_3 and so on. In data mining, rules could be extracted from data to describe exactly this probability. In recommender systems instead, it is not possible to infer good rules.

Hence, by using rules in a simple form, we only consider $P(\text{item } i|\text{item } j)$. With implicit ratings we can obtain

$$P(i|j) \approx \frac{\# < i, j >}{\# < j >} = s_{ij}, \tag{35}$$

which is a new definition of the similarity between item i and item j. It is an **asymmetric** similarity measure, since the similarity is different if the items i and j are switched.

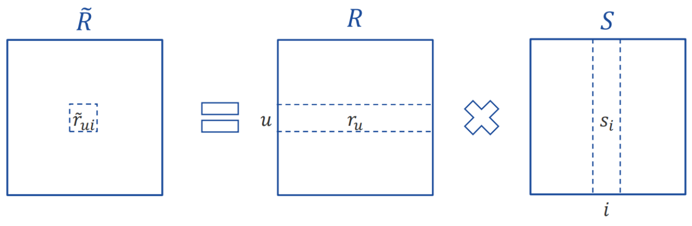
Also for association rules, it is necessary to add a shrink term C to the denominator of the conditional probability to avoid the user bias.

6 Item-based CF as Optimization problem

The item-based Collaborative Filtering technique can be seen as a machine learning optimization problem. It aims to estimate the rating a user would give to an item, based on the similarity with other items rated by that user. The simplest formulation is:

$$\tilde{r}_{ui}(S) = \sum_j r_{uj} s_{ji}, \tag{36}$$

corresponding to $\tilde{R}(S) = RS$ in matricial form.



Specifically, the similarity term can be computed in different ways depending on the similarity measure, so the estimated ratings are a function of the similarity matrix. $\tilde{r}_{ui}(S)$ and $\tilde{R}(S)$ can be defined as the machine learning model.

The estimated ratings are then compared with the real ratings to measure the quality of recommendations. This comparison is defined as the **Loss Function** $E(S)$, which is the objective function to be minimized by choosing the best similarity matrix. If $R = \tilde{R}$, $E(S) = 0$.

The core idea of a machine learning approach is to estimate the error of the model with a formula. Various options can be adopted for the loss function E:

- Error metrics such as MAE and MSE;
- Accuracy metrics such as Precision and Recall;
- Ranking metrics such as MAP and AUC.

6.1 SLIM

SLIM stands for **Sparse LInear Method**, and it is an item-based collaborative filtering approach based on machine learning.

Along with the definition of estimated ratings in Equation 36, here we focus on the objective function **MSE** defined as the square of the difference between the true rating and the estimated rating of a user:

$$E(S) = \sum_{u,i \in R} [r_{ui} - \tilde{r}_{ui}]^2. \tag{37}$$

In matrix notation we have:

$$E(S) = ||R - RS||_2. \tag{38}$$

The 2-Norm corresponds to compute the error over all the non-zero ratings in the URM, which we indicate as R^+. Hence, R^+ replaces R in Equation 37.

Then, the goal is to find the elements of the matrix S to reach the minimum of the 2-Norm:

$$S^* = \min_S ||R - RS|| = \min_S E(S). \tag{39}$$

If $S = I$ (I identity matrix), the error is reduced to zero, but this type of solution cannot generalize: it is unable to recommend items that a user never interacted with.

In order to avoid overfitting, it is necessary to:

- Force the elements on the diagonal of S to be zero.
- Add **regularization terms** to the error metric E.

  The minimization of E implies to solve the linear set of equations $\frac{\partial E}{\partial S} = 0$. Introducing the 2-Norm of the similarity matrix into E helps to keep the values in S as small as possible, in order to make the matrix sparse i.e. S only contains few non-zero elements. This version of modified error metric is called **Ridge Regression**:

$$E(S) = ||R - RS||_2 + \lambda||S||_2 \text{ , with } \lambda > 0. \tag{40}$$

In the two extreme cases, if $\lambda = 0$ we have $S = I$ with no regularization, and if λ is very large we have $S \to 0$ and no similarity is learned. In fact, λ is a **hyperparameter** that balances the minimization of the error and the reduction of overfitting represented by the regularization term. The process of finding a good value for λ is called **hypertuning**.

To reduce even more the risk of overfitting, the 1-Norm of the similarity matrix can be added to the loss function.

$$E(S) = ||R - RS||_2 + \lambda_A||S||_2 + \lambda_B||S||_1 \text{ , with } \lambda_A, \lambda_B > 0. \tag{41}$$

Now the regularization term is composed by both the 2-Norm and the 1-Norm of S, and both λ_A and λ_B are hyperparameters. This regularization is called **Elastic Net**.

6.2 BPR

Bayesian Probabilistic Ranking is another technique used to optimize the quality metrics of a recommender.

Instead of minimizing the MSE, here the goal is to optimize the **pairwise ranking**. Only implicit ratings are considered for simplicity. We use index i to identify relevant items, i.e. all the items the user has interacted with, and j for non-relevant items. If the estimated rating for i of user u is greater than the one for j, the estimated pairwise ranking will be identical to the true pairwise ranking.

So the objective function is to maximize the probability $P(\tilde{r}_{ui} > \tilde{r}_{uj} \,|\, u)$. The difference between the two values is defined as $x_{u_{ij}} = \tilde{r}_{ui} - \tilde{r}_{uj}$. The function σ - logistic sigmoid - is used to approximate that probability: if $x_{u_{ij}}$ is very large, the estimated probability is close to 1, otherwise it approaches 0.

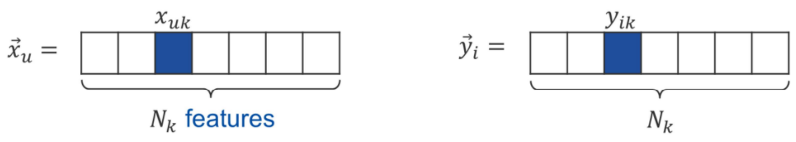
$$P(\tilde{r}_{ui} > \tilde{r}_{uj} \,|\, u) = \sigma(x_{u_{ij}}), \quad \sigma(x_{u_{ij}}) = \frac{1}{1 + e^{-x_{u_{ij}}}}. \tag{42}$$

In particular, the BPR algorithm optimizes the AUC, which is the area of the Fallout-Recall graph and should be kept as large as possible.

7 Matrix Factorization

Matrix Factorization is an algorithm based on Dimensionality Reduction, and it allows to improve existing Collaborative Filtering approaches by using a denser representation of the URM. Here the attributes of the items are called **features** or **latent factors**, for mathematical abstraction.

The user preferences for each feature are stored in the vector $\vec{x}_u$, with each element x_{uk} describing how user the user u likes the feature k. On the other hand, the item features are contained in the vector $\vec{y}_i$, and each element y_{ik} describes how much the feature k is important for the item k. Both vectors have length N_k, equal to the number of features used to described each item.



With Matrix Factorization, the rating for user u on item i is estimated as:

$$\tilde{r}_{ui} = \sum_k x_{uk} \cdot y_{ki} = \vec{x}_u \cdot \vec{y}_i. \tag{43}$$

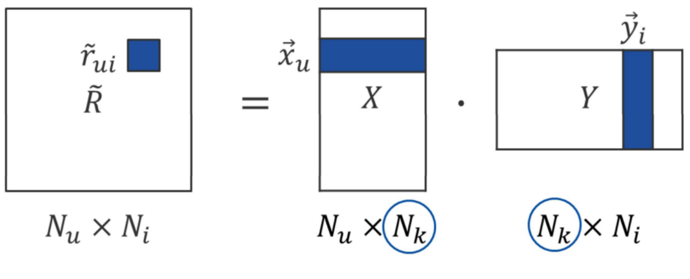
Expanding the computation for all the users and all the items allows to write the estimated ratings matrix $\tilde{R}$ as the product between the user-feature matrix X and the feature-item matrix Y:

$$\tilde{R} = X \cdot Y. \tag{44}$$

The matrices X and Y can be obtained using a machine learning approach. The ideal matrices X^* and Y^* are built by minimizing the error between the true rating and the estimated rating, and the simplest method is to adopt the MSE which relies on the Missing As Random assumption:

$$X^*, Y^* = min_{X,Y} ||R - \tilde{R}||_2. \tag{45}$$

In order to get good matrices X^* and Y^*, the number of latent factors - N_k - is a hyperparameter that needs to be tuned properly.



If the URM is very sparse, a large number of latent factors could lead to overfit the model to the few ratings available. Oppositely, if the URM is dense, a small number of latent factors reduces the personalization capabilities of the system. If $N_k = 1$, the system will recommend the most popular items. The ideal value for N_k generally falls into the range 10 - 200.

To prevent the risk of overfitting, regularization terms can be added in the error function to push the matrices X and Y to be sparse:

$$X^*, Y^* = \min_{X,Y} ||R - \tilde{R}||_2 + \lambda_1 ||X||_2 + \lambda_2 ||Y||_2. \tag{46}$$

When choosing the best feature matrices X and Y, the zeros in the URM can be considered in two different ways.

- **Missing-As-Random** is based on the 2-Norm and it is a naive assumption, but it works well if we are trying to estimate the real values of user ratings. It is adopted in Funk SVD.
- **Missing-As-Negative** is based on the Frobenius norm, which is the 2-Norm computed also on the zero elements. It should be used if we aim to optimize precision or recall. It is adopted in Bayesian Personalized Ranking.

7.1 Funk SVD

Funk SVD is a simple matrix factorization method, based on the **ALS** technique. It uses the model described in Equation 43.

Since the optimization (Equation 45) to find X^* and Y^* cannot be solved analytically, **Alternating Least Squares** has been proposed as a solution.

1. At the beginning, the matrices X and Y are filled with random values.
2. ALS minimizes two loss functions alternatively: at each iteration, one matrix is fixed while optimal values for the other one are learned, and vice versa.
3. The procedure continues until the improvements are irrelevant, or until the error is minimized.

In Funk SVD, the factors are not computed all at once.

1. At the beginning the number of latent factors $N_k = 1$, and ALS is applied to tune the matrices X and Y until the error is minimized.
2. After the convergence, we add an additional feature and repeat the process of tuning the matrices X and Y.
3. The operation of increasing N_k and tuning the matrices is repeated until the desired value of N_k is reached.

7.2 SVD++

SVD++ is an extension of Funk SVD. It introduces the global effects in the model used by Funk SVD:

$$\tilde{r}_{ui} = \mu + b_u + b_i + \sum_k x_{uk} \cdot y_{ki}, \tag{47}$$

with μ the overall average rating, b_u the user bias, and b_i the item bias.

SVD++ works well with a URM containing explicit ratings, since the global effects are able to normalize the estimated ratings.

Due to the introduction of the three scalar terms composing the global effects, it is not possible to use ALS to solve the optimization problem

$$\mu^*, b_u^*, b_i^*, X^*, Y^* = \min ||R - \tilde{R}||_2 + \lambda_1 ||X|| + \lambda_2 ||Y||. \tag{48}$$

So SVD++ typically adopts the **Stochastic Gradient Descent** technique.

However, even though SVD++ is an improved version of Funk SVD, these two techniques have the same disadvantage of not being not model-based: the model needs to be retrained every time a new user is added to the system, which means that the matrices X and Y, and eventually the global effects, should be to recomputed.

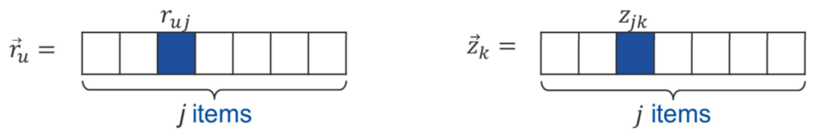
7.3 Asymmetric SVD

To avoid recomputing the estimated ratings from scratch every time a new user is added, the **Asymmetric SVD** method can be used.

In Funk SVD and SVD++, the vector x_u storing the user preferences must be computed for every new user. A solution to this problem is to approximate the value of x_{uk} as:

$$x_{uk} = \sum_j r_{uj} \cdot z_{jk} = \vec{r}_u \cdot \vec{z}_k, \tag{49}$$

where $r_u j$ indicates how much the user u likes the item j, and z_{jk} stands for how much the feature k is present in the item j.



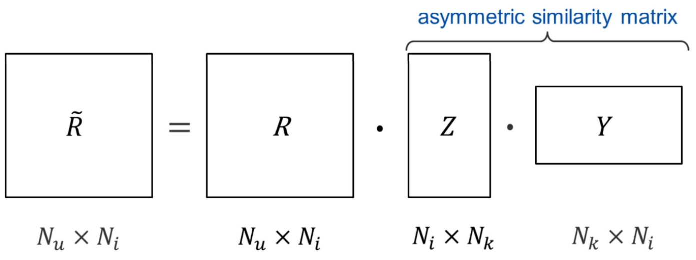
In this way, the technique allows to create a model by adding a vector that approximates the user preference. It is possible to give relevant recommendations to new users by asking for a few of their favorite items.

For a larger perspective, matrix X can be decomposed as the product of the matrices R (user ratings with respect to the items) and Z (importance of features in those items). The definition of $\tilde{r}_{ui}$ can be rewritten:

$$\tilde{r}_{ui} = \sum_k x_{uk} \cdot y_{ki} \quad \Rightarrow \quad \tilde{r}_{ui} = (\sum_k (\sum_j r_{uj} \cdot z_{jk}) \cdot y_{ki} = \sum_k \sum_j r_{uj} \cdot z_{jk} \cdot y_{ki}, \tag{50}$$

with r_{uj} the recorded user preference for item j, z_{jk} the latent feature preference factor, and y_{ki} how much the latent feature k is present in item i.

Now, $\tilde{R}$ is defined as the product of 3 matrices R, Z, and Y, and the product of Z and Y generates an asymmetric similarity matrix S.



Reintroducing the global effects in the calculation of the estimated rating, the formula becomes:

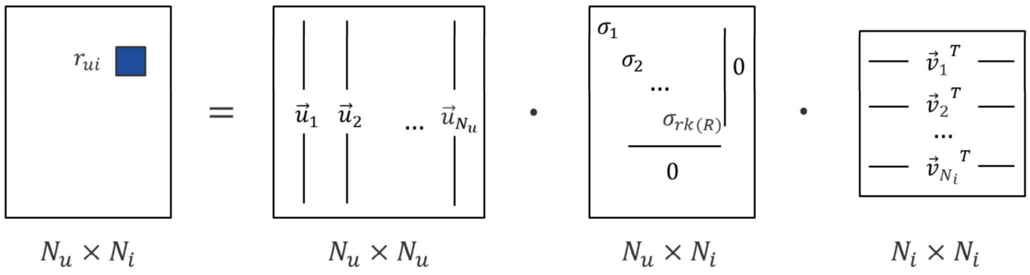
$$\tilde{r}_{ui} = \mu + b_u + b_i + \sum_k \sum_j r_{uj} \cdot z_{jk} \cdot y_{ki}. \tag{51}$$

7.4 Single Value Decomposition

This section presents the **Single Value Decomposition** method, which has inspired the previously mentioned techniques. SVD decomposes a matrix as the product of three matrices: U, Σ, V^T.

We apply the SVD method to the URM, i.e. the user rating matrix R:

$$R = U \cdot \Sigma \cdot V^T. \tag{52}$$



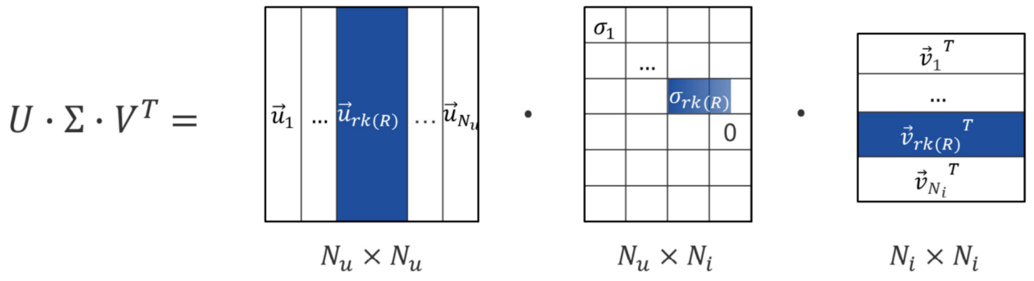
- R is the input data matrix with N_u users and N_i items. Each row $\vec{r}_u$ collects the preference of user u for all items.
- U is defined as left singular vectors. It is an orthogonal matrix: $U^T U = I, UU^T = I$, with rows corresponding to users and columns corresponding to latent features. Each cell defines how much a user is interested in a certain feature. The columns are eigenvectors hierarchically arranged by their ability to describe the variance in the columns of R, which means that the most important columns representing the matrix R are positioned on the left.
- Σ is a diagonal matrix of rank equal to the rank of R, and its diagonal contains the singular values of R. These values are non-negative and stored in descending order: $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_{rk(R)} \geq 0$. Each singular value indicates the strength of each feature in describing items and users in matrix R, σ_1 corresponds to the most important feature and so on.
- V^T is defined as right singular vectors, and it is orthogonal: $V^T V = I, VV^T = I$. The rows stand for latent features, while the columns stand for items. Hence, each cell defines how much a feature is important to describe a certain item. The rows are eigenvectors hierarchically arranged by their ability to describe the variance in the rows of R, and the most important rows representing R are positioned at the top.

7.4.1 Computation of SVD

A feature is computed at each step of the SVD decomposition in decreasing order of importance, for all the non-zero singular values:

$$R = U \cdot \Sigma \cdot V^T = \sum_{k=1}^{rk(R)} \sigma_k \, \vec{u}_k \, \vec{v}_k^T . \tag{53}$$

In fact, there can only be a certain number of linearly independent columns fixed by the rank, and all the other singular values are equal to zero nullifying the calculation.



7.4.2 Truncated SVD

If we consider separately the contributes from each feature, we have a sum of rank-1 matrices ordered by decreasing sigma values. The product $\sigma_1 \vec{u}_1 \vec{v}_1^T$ is called **best rank-1 approximation**:

$$\min_{\tilde{R}:rk(\tilde{R})=1} ||R - \tilde{R}||_F = \sigma_1 \vec{u}_1 \vec{v}_1^T. \tag{54}$$

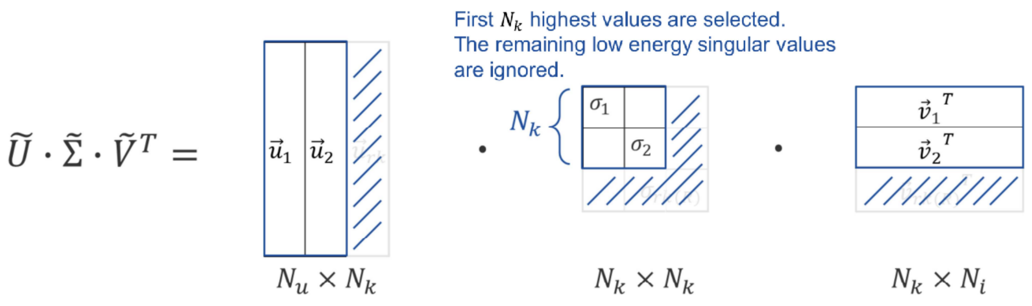
If also the contribution given by the second rank-1 matrix is considered and added to the first one, we will obtain the best approximation of rank-2 for R, and so on: the sum corresponds to a progressive approximation of R.

Truncation at rank $N_k \leq rk(R)$ is defined as the approximation of matrix R, calculated as the sum of the first N_k rank-1 matrices:

$$\min_{\tilde{R}:rk(\tilde{R})=N_k} ||R - \tilde{R}||_F = \sum_{i=1}^{N_k} \sigma_i \vec{u}_i \vec{v}_i^T. \tag{55}$$

In this way, R can be approximated as the product of three matrices $\tilde{U}$, $\tilde{\Sigma}$, and $\tilde{V}^T$, which are the truncated versions of the SVD decomposition matrices corresponding to the largest N_k singular values:

$$R = U \cdot \Sigma \cdot V^T \approx \tilde{U} \cdot \tilde{\Sigma} \cdot \tilde{V}^T. \tag{56}$$


First N_k highest values are selected.
The remaining low energy singular values are ignored.
$\tilde{U} \cdot \tilde{\Sigma} \cdot \tilde{V}^T =$
$\vec{u}_1$
$\vec{u}_2$
N_k
σ_1
σ_2
$\vec{v}_1^T$
$\vec{v}_2^T$
$N_u \times N_k$
$N_k \times N_k$
$N_k \times N_i$

It is important to notice that $\tilde{U}$ and $\tilde{V}^T$ are only column orthonormal:

$$\tilde{U}^T \tilde{U} = \tilde{V}^T \tilde{V} = I_{rk(R) \times rk(R)}, \tilde{U}\tilde{U}^T, \tilde{V}\tilde{V}^T \neq I. \tag{57}$$

SVD works as a recommender algorithm only if it truncated, since computing the full SVD of the URM corresponds exactly to the original URM, and we obtain zero if we want to estimate missing ratings. When used for recommendations to predict missing ratings, the Truncated SVD technique is called **Pure SVD**.

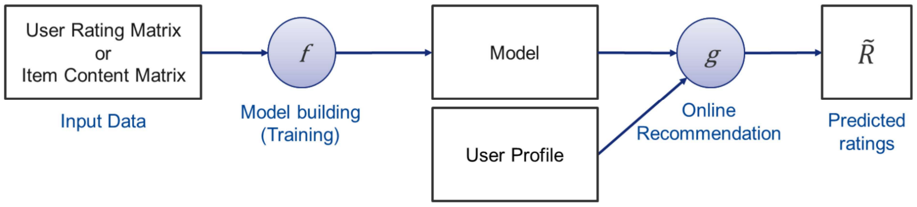
$$R \approx \tilde{R} = \tilde{U} \cdot \tilde{\Sigma} \cdot \tilde{V}^T \tag{58}$$

The newly obtained matrix $\tilde{R}$ approximates R. It keeps track of the main features, and at the same time it helps to reduce overfitting.

8 Hybrid Recommender Systems

Based on the idea of taking advantage of the strengths of various classes of recommender systems, the result of combining different component algorithms into one is called **hybrid recommender system**.

In a generic model-based recommender system, the algorithm builds a model from the input data containing all the relevant information required to later provide recommendations. When using machine learning approaches, this first phase is called the training of the model. After that, the model can be used to estimate ratings and to generate real-time recommendations for any user. The only requirement is to provide as additional input the user profile, i.e. the past ratings of a user.

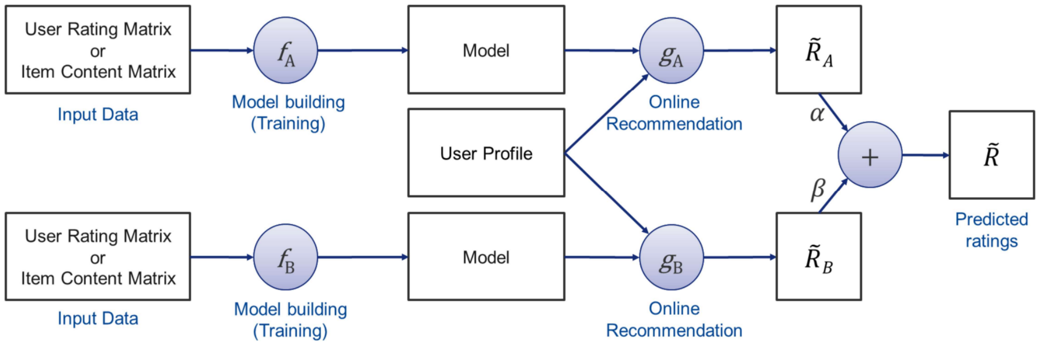


Many different approaches can be used to build a hybrid recommender system, and they can be roughly classified into 5 families: Linear Combination, List Combination, Pipelining, Merging Models, Co-Training.

8.1 Linear Combination

A natural way to combine the recommendations of two algorithms consists in adding their predicted ratings together by means of a **linear combination**.

The ratings for a specific user are estimated using the algorithms A and B in an off-the-shelf fashion (without any modification) respectively, and the weighted sum between the two outputs is computed. This construction allows an easy combination of the algorithms and their parallel execution.



The linear combination can be represented in matrix notation, or in terms of the individual entries of the matrix:

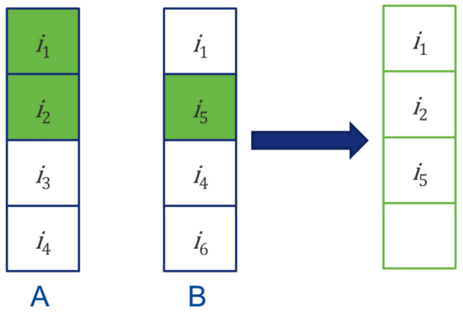
$$\tilde{R} = \alpha \cdot \tilde{R}_A + \beta \cdot \tilde{R}_B, \quad \tilde{r}_{ui} = \alpha \cdot \tilde{r}_{ui}^{(A)} + \beta \cdot \tilde{r}_{ui}^{(B)}. \tag{59}$$

Linear combination is very straightforward to implement, but the problem is that the optimal values of the weights α and β are different for each dataset, and these hyperparameters must be carefully tuned in order to have an effective hybrid system. Moreover, if the estimated ratings are on different scales, it will be more difficult to obtain good results. In this case, it is particularly important to weight the different components properly.

8.2 List Combination

This method is based on the combination of the recommendation lists. Assuming to have two recommendation lists produced by two different Top-N algorithms respectively, it is possible to take only the highest ranked items from each list and merge them: hence, a new Top-N list is generated.

One simple yet effective approach to merge two ranked lists is to use **round-robin**: items are selected starting from the top of each list in turn, in a "circular" way. If the two lists share some items, we ignore the items that we have already considered, and go back to the other list for the selection.

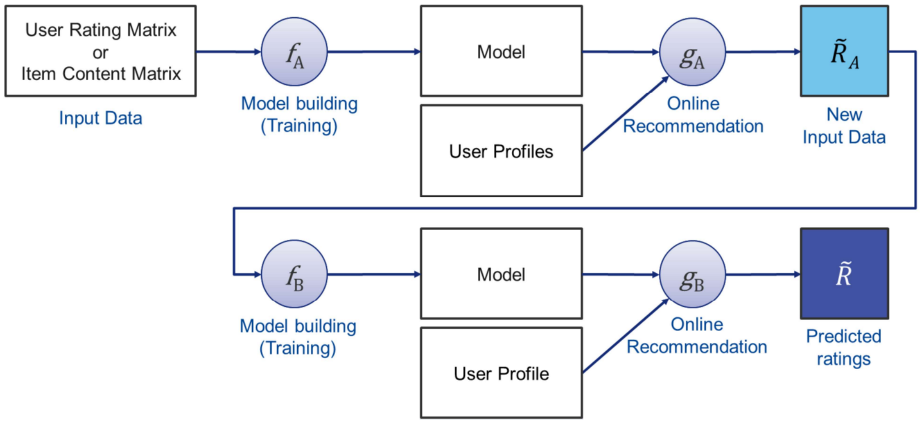


Also list combination makes use of the component algorithms as-is, in an off-the-shelf fashion. The advantage of this method is that it does not require comparable rating scales between its component algorithms. On the other side, it is difficult to weight the relative importance of each list.

8.3 Pipelining

Pipelining consists in chaining two or more algorithms, such that the output ratings of one algorithm is fed as input to the next algorithm in the pipeline.

For example, considering two recommender algorithms A and B, the ratings estimated with A can be used to enrich an existing URM, which becomes the input for B. The output of A must be computed by using the profiles of the users taken from the URM that we wish to enrich for the algorithm B. The overall result is the final matrix of estimated ratings.

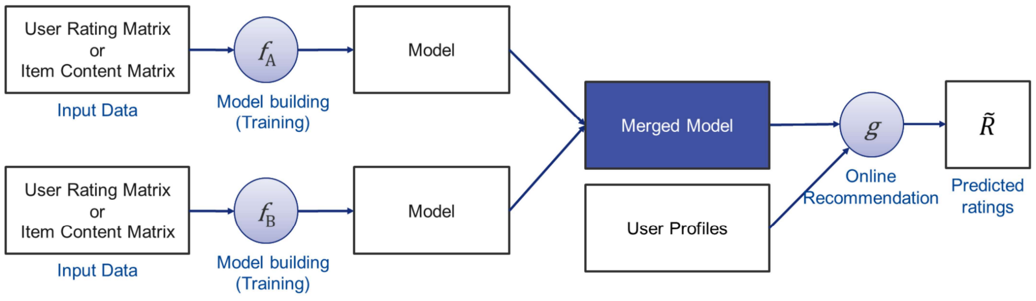


A practical application of pipelining is to use a Content-based Filtering algorithm as the first stage with an input ICM, to enrich a URM that is later used by a Collaborative Filtering algorithm.

This approach increases the density of the URM. Although this is beneficial in terms of quality of recommendations, it can be problematic in terms of memory and computational requirements. So we must carefully decide how much to enrich the intermediate URM.

8.4 Merging Models

A hybrid algorithm can be built by merging two or more models, but only if these two are compatible: they are required to have the same data structure. It is not possible to merge a similarity matrix of a collaborative algorithm with the factor matrices of a matrix factorization algorithm, nor to merge the user similarity of a user-based CF algorithm with the item similarity of a CBF algorithm.



We consider as the example a CF item-based algorithm A with an input URM, and a CBF algorithm B with an input ICM. Both algorithms produce an item-item similarity matrix as their model, so they can be merged: the two output similarity matrices S_A and S_B are combined by means of a linear combination to obtain the hybrid similarity matrix S:

$$S = \alpha S_A + (1 - \alpha) S_B, \tag{60}$$

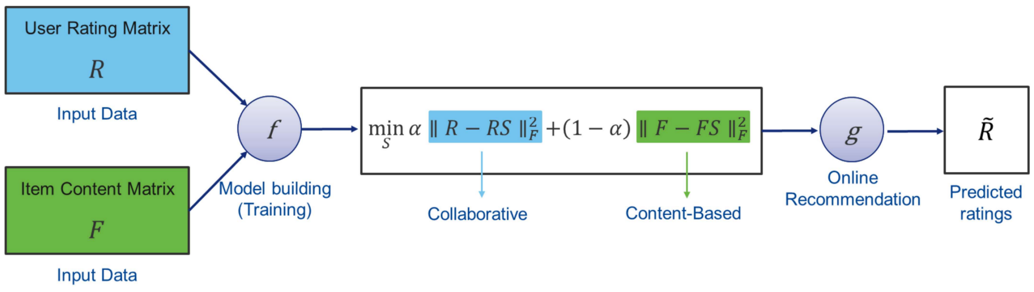
with α a hyperparameter that must be tuned. At this point, the ratings are estimated using the new model.

8.5 Co-Training

This technique belongs to the family of approaches called **Collaborative Filtering with Side Information**. **Co-Training** approaches are similar to the Merging Models method, but in this case the algorithms A and B must be trained together, and the merged model is obtained directly during the training phase.

8.5.1 S-SLIM

S-SLIM (SLIM with side information) is reported as an example of Co-Training: it combines both the CF and the CBF approaches.



For simplicity, in the formula

$$\min_S \alpha ||R - RS||_F^2 + (1 - \alpha)||F - FS||_F^2 \tag{61}$$

we ignore the regularization terms and other constraints (such as the diagonal of S being zero), but they are necessary for good estimations.

So the optimization problem is composed of two parts:

- Collaborative part: it represents the basic SLIM algorithm, while builds the model from the URM R by optimizing the item similarity matrix S. The error between the true and the estimated ratings is minimized: the Frobenious norm (as in the equation) or other error metrics, such as pairwise ranking, can be used.

- Content-Based part: the ICM F is considered. Based on the idea of predicting missing attributes for an item by looking at the attributes of the similar items, it is possible to build a content-based similarity matrix using machine learning, by minimizing the error between the true ICM and the predicted ICM.

Hence, the optimization formula for S-SLIM includes both the URM and the ICM. In this way, the matrix S to be learnt must fit both of them. A coefficient α is used to weight the two parts: $\alpha = 1$ implies a pure collaborative filtering technique, while $\alpha = 0$ indicates a content-based only approach. If the value is between 0 and 1, we get a hybrid model. Techniques like grid search can be used to find the best value for α.

The new model exploits the correlation that exists between the users' behaviour on two items, and the similarity of the two items' intrinsic properties. The items' attributes are called **side information**, and the method takes its name from here.

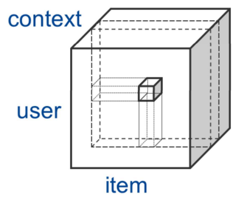
9 Collaborative Filtering with Side Information

The family of algorithms based on collaborative filtering with side information includes the previously mentioned Co-Training method along with many other approaches.

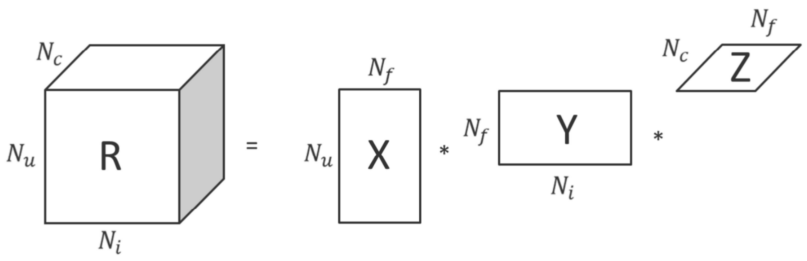
9.1 Context-Aware Recommender Systems

Context-Aware recommender systems are another example of collaborative filtering with side information. They include, as an additional information, the context in which a user interacted with an item.

The easiest way to build a context-aware recommender system is by using **tensor factorization**. A **tensor** is a matrix with 3 or more dimensions, and tensor factorization extends matrix factorization by adding the **contextual dimension** to the URM. So now the three dimensions are the user, the item, and the context respectively.



For simplicity, we consider an implicit URM and a matrix factorization model without global effects. The URM R is factorized into the matrix of user factors X and the matrix of item factors Y. N_u represents the number of users, N_i the number of items, and N_f the



number of latent factors. With the tensor factorization, X and Y are multiplied by a third matrix Z representing the contextual factors. N_c is the number of contexts.

The rating for user u on item i in context c can be estimated as the sum of the products of the corresponding three factors:

$$\tilde{r}_{uic} = \sum_f x_{uf} \cdot y_{if} \cdot z_{cf}. \tag{62}$$

In matrix format, the approximate rating tensor is computed as the product of the three matrices:

$$\tilde{R} = X \cdot Y \cdot Z. \tag{63}$$

In order to find the tensor factorization, we can minimize a loss function computed as the MSE between the true ratings and the estimated ratings, plus the regularization terms:

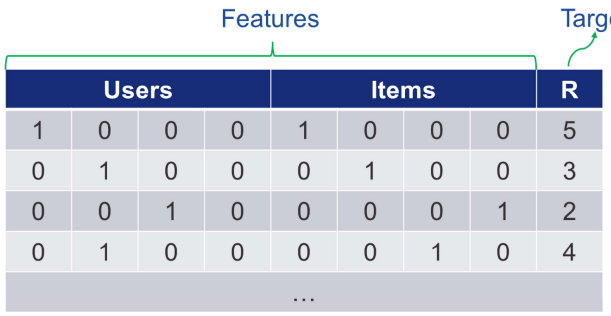
$$\min_{X,Y,Z} ||R - \tilde{R}||_2 + \lambda_1 ||X|| + \lambda_2 ||Y|| + \lambda_3 ||Z||, \tag{64}$$

where the regularization terms are required to keep X, Y and Z as sparse as possible to avoid overfitting and to increase the ability of the model to generalize.

9.2 Factorization Machines

Factorization Machines represent a technique of collaborative filtering with side information typically used in classic data mining problems.

9.2.1 Input data representation


Features
Target

Users				Items				R
1	0	0	0	1	0	0	0	5
0	1	0	0	0	1	0	0	3
0	0	1	0	0	0	0	1	2
0	1	0	0	0	0	1	0	4
...								

The input to factorization machines is structured as a table, where all the columns - except the last one - are divided into **groups**. Each group correspond to one dimension of the URM, and these columns represent the **Features** of the dataset. In the example we have two groups: Users and Items.

The users and the items are represented through One-Hot Encoding: by assigning a column to every user and item, we can specify a certain user and a item by setting the proper column to one and all the others to zero. So, each row represents an interaction between a certain user and a certain item, and the last column in the table contains the ratings for those interactions. This final column is our **Target**.

In this way, the same information of the URM is represented, but this format uses a larger table and it is more sparse: each interaction in the URM corresponds to a row in the table.

9.2.2 Model components

For now we assume non-zero interactions, but also zeros can be inserted into the table. The model used to estimate unknown ratings is described by:

$$\tilde{r}^{(k)} = \omega_0 + \sum_{i=1}^{n} \omega_i \cdot x_i^{(k)} + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \omega_{i,j} \cdot x_i^{(k)} \cdot x_j^{(k)}, \tag{65}$$

with the index k specifying the row, and the indices i and j indicate columns. n is the number of columns, equal to the sum between the number of users and the number of items. $\tilde{r}_k$ is the rating that we want to estimate for row k. $x_i^{(k)}$ is the element of the table at row k and column i, and it can be either 0 or 1.

The model is composed of three components:

- Constant component: ω_0. It is the intercept of the model function, and it is a scalar number representing the global bias. If only ω_0 is used in the model, it is equal to assuming that all ratings are identical to a constant value.
- Vector component: $\sum_{i=1}^{n} \omega_i \cdot x_i^{(k)}$, it is a linear combination of the variables. ω_i is an element of a vector whose dimension is n, and it models the strength of the i-th input variable (feature). Essentially, a bias is chosen for a user and an item that are specified in the interaction.

  At this point, the estimated rating for each row is composed by three non-zero terms:

  $$\tilde{r}^{(k)} = \omega_0 + \omega_u + \omega_i, \tag{66}$$

  which is related to the global effects $\mu + b_u + b_i$.
- Matrix component: $\sum_{i=1}^{n} \sum_{j=i+1}^{n} \omega_{i,j} \cdot x_i^{(k)} \cdot x_j^{(k)}$. This last component is equivalent to matrix factorization: the relation between two features appearing together in one interaction is represented in a matrix of weights $\omega_{i,j}$. The matrix is assuming to be

square and symmetric: $\omega_{i,j} = \omega_{j,i}$. In our example, interactions are only between users (i) and items (j), and $\omega_{i,j}$ represents that component of the rating.

The overall estimated rating is the sum of the global effects and a matrix component that acts as the local effect, by modelling how much a specific user likes a specific item. The index j starting at $i+1$ in scanning the columns of features allows to find two distinct features that are both present in that interaction.

9.2.3 Parameters

The values (ω_0, ω_i, $\omega_{i,j}$) are parameters that we want to learn. They do not depend on k, which means that they are the same for all the interactions. The learning algorithm must find these values in such a way to reproduce the ratings for every interaction as accurately as possible.

To learn the parameters omega, a loss function between the true ratings and the estimated ratings needs to be minimized:

$$\text{argmin}_{\omega} L(r^{(k)}, \tilde{r}^{(k)}) + \lambda ||\omega||. \tag{67}$$

Any loss function can be used in this case, and the regularization term could be introduced for better performance.

The factorization machines model can be represented by using a vector and matrix notation:

$$\tilde{r}^{(k)} = \omega_0 + \omega \cdot \mathbf{x}^{(k)} + \mathbf{x}^{(k)'} \cdot \mathbf{W} \cdot \mathbf{x}^{(k)}, \tag{68}$$

where $\mathbf{x}^{(k)}$ is the vector of features, ω_0 is the unknown global bias, ω is the vector of unknown biases (size n), and $\mathbf{W}$ is the matrix of unknown coefficients ($\frac{n^2}{2}$ values, since it is symmetric). So the overall number of parameters to be learned is $1 + n + \frac{n^2}{2}$.

9.2.4 Dimensionality problem

In an actual dataset, the total number of ratings is much smaller than the total number of interactions. Since the matrix $\mathbf{W}$ is huge and very sparse, we aim to reduce its dimension with matrix factorization technique that is common in data mining.

The matrix $\mathbf{W}$, sized $n \times n$, can be approximated as the product of two rectangular matrices A and B: A has dimensions $n \times k$, and B is $k \times n$. k is called the number of latent factors, and $k \ll n$. If A^T is equal to B, the matrix $\mathbf{W}$ would be symmetric.

$\omega_{i,j}$ is approximated as the dot product between the row i of the matrix A, and the column j of the matrix B.

$$\omega_{i,j} = \mathbf{a}_i \cdot \mathbf{b}_j = \sum_{h=1}^{k} a_{i,h} \cdot b_{h,j}. \tag{69}$$

This can also be expressed as a summation of the product of $a_{i,h}$ and $b_{h,j}$, essentially it is the application of matrix factorization to the last set of parameters only.

After the **dimensionality reduction**, the rating equation becomes:

$$\tilde{r}^{(k)} = \omega_0 + \sum_{i=1}^{n} \omega_i \cdot x_i^{(k)} + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{h=0}^{k} a_{i,h} \cdot b_{h,j} \cdot x_i^{(k)} \cdot x_j^{(k)}. \tag{70}$$

As a result, the matrix component of the equation is composed of three summations. Instead of having a $n \times n$ matrix, we have A and B each having $n \times k$ parameters. So the overall number of parameters is now reduced to $1 + n + 2nk$ ($1 + n + nk$ considering $B = A^T$).

It can also be expressed in terms of vector notation and matrix operations:

$$\tilde{r}^{(k)} = \omega_0 + \omega \cdot \mathbf{x}^{(k)} + \mathbf{A} \cdot \mathbf{B} \cdot \mathbf{x}^{(k)} \cdot \mathbf{x}^{(k)}. \tag{71}$$

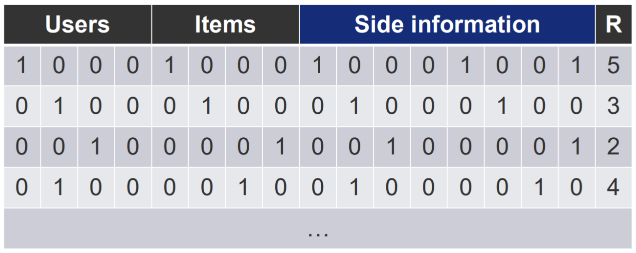
9.2.5 Comparison with matrix factorization

Despite the similarity of the name, matrix factorization and factorization machines are two different techniques.

In general, factorization machines are more powerful than matrix factorization, because we are not bound to have only one user and one item per interaction: also group preference can be considered to provide a better recommendation. Also, when modelling group interactions, the importance of some users and some items can be reduced by assigning values of x_i smaller than 1.

9.2.6 Comparison with CF techniques

The Factorization Machines formula would identical to collaborative filtering matrix factorization, if there is only one user and one item for every interaction. However, FM are better than classical CF techniques for their flexibility. They can be optimized for various applications by simply adding columns containing different kinds of side information to the input table, without changing the model itself. Potential side information could be the context, the attributes on items, or the attributes on users.



Users				Items				Side information								R
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	5
0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	3
0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	2
0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	0	4
…																

If we integrate the dataset with content information, it is possible to exploit a mixed collaborative content-based technique.

9.2.7 FM with implicit ratings

Using implicit ratings in factorization machines will cause the **imbalance** problem, since there is no information on negative interactions and the algorithm would learn to rate every interaction with the value of one.

Hence, there is need to create a balance between positive and negative samples by adding an equal number of negative interactions. We can randomly choose some of the missing ratings and insert them into the table, despite that we do not know if the user has ever interacted with them or did not like them. Actually, the fact that the user only likes a small subset of any real dataset allows to assume missing ratings as negative, so there would be a high probability that the user would not like an item chosen randomly.