# Tutorial of Recommender System
-
# Advanced Statistical Learning

Guangmei Ye(11254651) Miao Hu (11255049)

March 2020

## 1 Presentation of the problem

### 1.1 Background for the generation of the recommender system

With the rapid development of the Internet, we have entered the era of information explosion. At present, more and more platforms provide services through the Internet. The corresponding types of services (shopping, video, news, music, marriage, social networking, etc.) are emerging endlessly. And the kinds of subjects of services are becoming more and more diverse. How so many "subjects" make it needed by people who need it to meet the various needs of users is a difficult problem facing enterprises.

At the same time, with the development of society and the improvement of education, everyone has the desire to express their individuality. Since the introduction of the Internet, there have been a lot of products that can express their personalities, such as Facebook, Instagram, Twitter, etc. Each person's personality and preferences have a great space for display. Also, from the perspective of evolution, each person is a differentiated individual, born differently, born with different personality characteristics, and the intimate environment of life and growth is much different, resulting in individual preferences and tastes. The "Long Tail Theory" also explains very well that the non-selling items in diversified items can meet people's diverse needs, and these needs do not necessarily add up to fewer sales than popular items.

With the progress of society and the improvement of material living conditions, we no longer need to worry about survival. Therefore, more and more of our needs are non-survival needs, such as reading books, watching movies, shopping, etc., and these non-survival needs are often In many cases, it is uncertain, unconscious, and people don't know what they need. The need for survival seems very strong and evident to people. For example, if people are starving, the first need must be food. Different from survival needs, in the face of non-survival needs, people are more willing to accept passively recommended useful items, such as supporting a movie to you. If it meets their taste, they may like it very much.

To sum up, the three points mentioned above, there are so many goods and services to choose from in today's era, and the interests and preferences of different people are very different. In specific scenarios, individuals' needs for themselves are not very clear. Driven by these three backgrounds, the recommender system came into being. The personalized recommender system is one of the most effective methods and tools to resolve the above three contradictions.

### 1.2 The common obstacles in recommender systems

The recommender system satisfies the needs of the provider, platform, and user. The recommender system can better expose the products to users who need to purchase and improve the efficiency

of the allocation of social resources. In essence, the recommender system improves the efficiency of information distribution and information acquisition. The recommender system is very valuable, and it is worthy of spending much energy and time to build a good recommender system. To better serve users and earn more profits while providing users with services, more and more companies are adopting personalized recommender technology to help users discover what they like faster than before. However, it is difficult to build an efficient and valuable recommender system. Here is a brief description of the challenges of building a good recommender system.

(1) Cold start problem. Cold start is generally divided into new user cold start and new "target" cold start. For a new user, because there is no relevant behaviour or very few behaviours, the user's interest preferences cannot be obtained, so he cannot make effective recommendations for him. For the newly-launched / online subject, since there are no users or few users operate it (click, browse, comment, purchase, etc.), we do not know what type of users like it, and it is difficult to make good recommendations. (2) Data sparsity. Due to the huge number of "target objects" involved in many recommended application scenarios, such as tens of billions of articles, tens of millions of products, etc., many data users behave sparsely. For the same "subject," only a few users have related behaviours, which makes it very difficult to build a recommender algorithm model. (3) Matthew effect. Head objects are consumed by more and more users, while good-quality long-tail objects are not given enough attention due to less user behaviour and insufficient self-descriptive information. (4) Gray sheep effect. It means that the tendencies and preferences of some users are not obvious, they are scattered, and they do not show a strong preference for objects with certain characteristics. Therefore, the gray sheep effect in the collaborative filtering recommender algorithm is obvious. For example, based on the user's collaborative filtering, the similarity of such users with low preferences is similar to that of other users. There is no difference in selecting different similar users, so the recommendation effect is not particularly good. This problem is very obvious when multiple users use the same device. For example, a family uses the same TV to watch their favourite content at different times, resulting in a wide range of recommended behaviours on the TV without any characteristics. (5) Portfolio effect. Since the objects obtained from different channels are very similar, the recommender system may recommend very relevant objects. But for users, these related objects are duplicated and worthless. This situation often occurs in news information and short video app recommendations. For example, multiple media channels report on the same hot event, and the content may be similar and repeated. The system did not identify the content in the process of storing it in the database. Therefore, the content is considered different Content. The recommender system makes it easy to recommend them to users together. (6) Stability/plasticity issues. This problem refers to the fact that when the user's interest stabilizes, it is difficult for the recommender system to change the user's perception. Even if the user's interest has changed recently, the recommender system still retains the user's previous interest. Unless the user's new interests accumulate enough, the role played by the new interests completely overrides the old ones. In general, the idea of solving this problem can perform a time decay operation on user interest. The recent behaviour weight is large, and the older the behaviour weight is, the smaller the behaviour weight is.

In this tutorial, we will go through different algorithms or models in the recommender system topic, and we will present their theoretical basis, what packages are available on R, what are those R packages capable of and then present how to implement them in R.

## 2 Literature review of literature on this subject

Recommender systems collect information about preferred people-users on a set of items (such as movies, songs, books, applications, websites, travel destinations, and e-learning materials). Information can be obtained explicitly (usually by collecting user ratings) or implicitly (usually by monitoring user behaviour (such as songs you have heard, applications), downloads, websites you have visited, and books you have read).

There are many recommendation strategies in this field[1], including:

- Collaborative Filtering(CF): This strategy uses the rating data among users to find different groups of users that are similar in terms of ratings to give recommendations based on these similarities among users groups. This strategy assumes that users who gave similar ratings to the same item will also give similar ratings to the same item in the future.
- Content-Based Filtering(CBF): This strategy uses the features of items and compares them with users' past behaviours to give a recommendation. It assumes that users who give high ratings to items with certain features will do the same in the future.
- Demographic Filtering(DF): This strategy aimed at using demographic data about the users, including age, gender and etc. to identify the user groups and give recommendations based on it. It assumes users within the same group will behave similarly.
- Knowledge-Based Filtering(KBF): This strategy uses features of items and also the knowledge of how the item meets the users' needs to provide recommendations.

The history of the Recommender system can be traced back to the early 1980s when a word-vector based algorithm for searching amongst textual documents was introduced. Throughout the years, these algorithms were expanded to accept different types of content, including documents, emails and multimedia items in a literature survey for Research-paper recommender systems posted in 2016. J.Beel et al. (2016)[2] mentioned that in the last 16 years, more than 200 research articles were published about recommender systems. More than half of the researchers applied Content-based filtering (55%). Collaborative filtering was applied by only 18% of the reviewed approaches and graph-based recommender by 16%. Other recommender concepts included stereotyping, item-centric recommender, and hybrid recommender

In recent years, deep learning in the recommender system field is full of innovations; huge numbers of deep learning recommender systems have been proposed in different publications. There are several strengths for applying deep learning techniques to this problem, including non-linear transformation and so on. As traditional recommender system techniques have linear assumptions and oversimplify the problem, using deep learning techniques might have some breakthroughs. Also, according to S. Zhang et al. (2019), there are several bases deep learning techniques are being used, for example, CNN, RNN, Autoencoder, Neural Autoregressive Distribution Estimation, Restricted Boltzmann machine, multi-layer perceptron, Neural Attention, Adversary Network and Deep Reinforcement Learning.

Additionally, hybrid methods that combine two or more models listed above are also very popular. Usually, using only one kind of strategy has some disadvantages. For example, Collaborative Filtering and Content-Based Filtering suffer from the cold-start problem, Demographic Filtering suffers from limited databases, and so on. Thus it's reasonable to use a hybrid system that combines two or more strategies. For example, using Collaborative Filtering with KBF or DF, or using CF with CBF or with additional DF and so on. There are many different combinations of hybrid systems; sometimes, there are combinations of other data mining or machine learning techniques included in those combinations.

There are also other algorithms that were not specifically designed for this area and CF. However, researchers have found different use cases for these methods, including Principal Component Analysis and K-Nearest Neighbours. For example, Goldberg et al. (2001)[3] proposed a collaborative filtering algorithm that combines clustering and PCA called Eigentaste and B. Wang et al. (2017)[4] purposed a weighted K-NN which use Item Rating-Inverse User Frequency which is based on the algorithm of TF-IDF (Term Frequency-Inverse Document Frequency)

There are several bottlenecks for the current research in this field. First of all, it is still unclear which recommender technique is the best. For example, researchers have reported different results on content-based filtering and collaborative filtering; some researchers found CBF is better than CF, while other researchers found the opposite conclusion. However, there are three reasons behind this situation: Firstly, many kinds of research have their limitations as they implement different techniques on strongly pruned datasets or too few participants in the studies or not using proper baselines algorithms. Secondly, many algorithms implemented in those research papers are too difficult to be implemented

again, as the authors of this research provided too little details on their algorithms. Thus it's difficult to reproduce the results given by the researchers. Thirdly, even a minor variation in the dataset and algorithms would create a big variation in the performance of the algorithms implemented. As for the second bottlenecks, many researchers tend to ignore all factors like user satisfaction rate on the recommendations except for accuracy. Lastly, few papers actually have an impact on the recommender system field, and also most of the authors who have published a paper in the recommender system area wouldn't publish another paper in this field.

There are several popular databases available for this research field, including MovieLens, Filmtrust, Epinions, Yelp, KB4Rec and also Amazon Review Database from UCSD.

# 3 Brief description of the methods

## 3.1 Non-personalized systems

Non-personalized systems, just as the name suggested, are the simplest type of recommender systems, as they don't take personal features of the user into account. As a result, those recommender systems will produce identical recommendations to all customers; these recommendations are just either a list of items that users, on average, would like. There are two main types of algorithms involved with Non-personalized systems: Aggregated Opinion Approach and the Basic Product association.[5]

### 3.1.1 Aggregated Opinion Approach

On movie websites or on an e-commerce website, you would probably encounter reviews that viewers or consumers made along with one or more scores. These scores generally have a range from 1 to 5 with an increment of 0.5. On those websites, you would also find a part saying "Movies you might like" or "Something you might like" with average scores. The average score calculated for those scores are just simply calculated as

$AverageScore = Mean(score * 10)$

This approach is good to capture the overall population's preference; it can give Top-N movies or top-N restaurant in this area etc.[6]

### 3.1.2 Product Association

This method, compared with Aggregated Opinion Approach, considers the context. For example, you could see "People who watched movie A also watched movie B" when you are viewing the description of Movie A or after you watched Movie A. In this situation, the recommender system is still non-personalized as it doesn't consider item-based or user-based features, however, it does consider the user's behaviour. The idea for this system is that: we know the fact that people who did A and also did B or C or D. We calculate the percentage of consumers who bought A also bought one of B, C and D, and then rank them to get the highest percentage and make a recommendation based on it.

## 3.2 Collaborative Filtering

As we all know, one of the most classic technologies in the recommender system is Collaborative Filtering(CF), which is based on the assumption that if a user has been interested in certain projects in the past, he will likely still maintain enthusiasm for them in the future. Among them, the collaborative filtering technology can be divided into memory-based CF and model-based CF based on whether or not machine learning ideas are used for modelling.

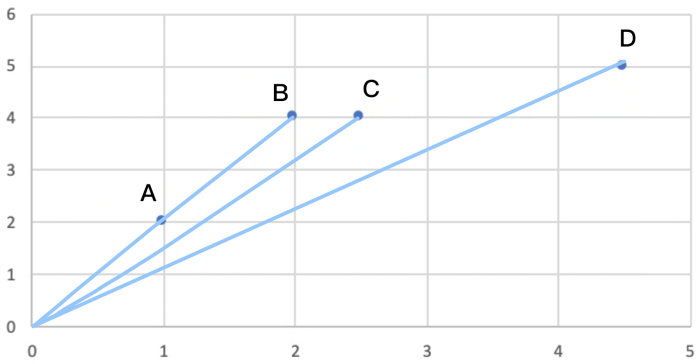
### 3.2.1 Memory-Based CF

Collaborative filtering is based on historical data. It assumes that user preferences tend to be consistent from past to future. Thus, it requires no more information other than the clients' historical

preferences for sets of products. There are two categories that can represent user preferences. One is the explicit rating, which is the most direct feedback from the user on the product to measure how much they like the product. For example, the user gives a rating on a product, such as the rating of each product in Amazon. The other is the implicit rating. It indirectly determines user preferences based on page views, clicks, purchase history, and more.

There are two steps to implementing collaborative filtering. First, we need to find users similar to U who have related the item I. Then, we can calculate the rating R based on the rating of users found in step one.

Many ways can be applied to find similar item groups. The simplest way is to measure the Euclidean distance. The smaller the distance, the more similar. A better way is to calculate the Cosine distance. When we connect each point with the zero, we can capture the angle between different data points. The larger the angle, the less similar. For example, in the picture following, users A and B are considered absolutely similar in the cosine similarity metric despite having different ratings and large Euclidean distance. This is actually a common occurrence in the real world, and the users like the user A are what you can call tough raters. We can further adjust the ratings to give an average of 0 for all users, which brings them all to the same level and removes their biases. This is called centered cosine.




After we have found a set of users similar to a user U, we need to calculate the rating R that U would give to a certain item I. Again, we can achieve this in various ways. We can use the average of the ratings for the top 5 or top 10 users most similar to U. In some situations, the similarity users we found do not have equal similarity to the user U. In this case, we can use the weighted average by using the similarity factor as weight. In this function, every rating is multiplied by the similarity factor S of the user who gave the rating. The final predicted rating by user U will be equal to the sum of the weighted ratings divided by the sum of the weights.

$$R_U = (\sum_{u=1}^{n} R_u * S_u)/(\sum_{u=1}^{n} S_u) \tag{1}$$

Normally, Collaborative filtering (CF) is mainly divided into user-based and item-based. The difference is in step one if the model finds similar users based on the ratings they gave by items, it is called User-based CF. Inversely, if the model finds similar items based on the rating they gave by users, it is Item-based CF. The item-based algorithm was developed by Amazon. In a system with more users than items, item-based CF is faster, more stable, and more effective than user-based CF. Because generally, the average rating of a product does not vary as much as a user's rating of different products. So, when scoring the matrix coefficients, it has a better effect than the user-based method. However, for the databases related to browsing and entertainment, the Item-based method performs poorly. This type of data usually has better results using Matrix factorization or hybrid recommenders. A good data recommender system is usually built by combining several different algorithms.

Based on the historical data between users and items, we will get a two dimensions matrix, including the number of users and the number of items. Generally, each user will only mark a part of the items.

Thus this matrix will exist with a lot of empty space. Sometimes, the matrix can be very sparse. It is a common and important problem for collaborative filtering. If we fill the empty space with a random value, it will cause inaccuracy and change the characteristics of these users. If we replace missing value by average rating, the tough raters and not tough raters who have the same ratio will be recognized as totally dissimilar users. A reasonable solution is that by using a centered cosine, also called Pearson correlation; each user can adjust the value to make the centered average of scores equal to 0. At this time, using zero to fill the empty value will not change the original characteristics of the matrix. In addition, in real life, there are many specialized algorithms for this problem. We will introduce them in detail below.

### 3.2.2 Model-based CF

When the matrix is mostly empty, reducing or compressing the dimension of the matrix can boost the algorithm efficiency in both time and space. We can use Matrix factorization or autoencoders to operate. Matrix factorization is a method to break down a large matrix into a product of the smallest ones. The number of reduced matrices can be more than two as well. The reduced matrices actually represent the user and items individually. Thus, when the model automatically plays their part to analyze data and complete matrix factorization, it also can dig out the insights about users and items, finding the latent factors behind the data. The greater the number of factors, the more personalized the recommendation becomes. But too many factors can also lead to overfitting.

Before introducing the large family of matrix factorization, let us clarify the scenario of the recommender system and the principle of matrix factorization. For the recommender system, there are two major scenarios: rating prediction and Top-N recommendation (item recommendation, item ranking). The rating prediction scenario is mainly used to evaluate websites, such as how many users rate movies (MovieLens) they have watched, or how much they rate books (Amazon Kindle). The matrix factorization technique is mainly used in this scenario. The Top-N recommendation scenario is mainly used for shopping websites or websites that do not generally receive explicit rating information, that is, to recommend a list of possible interests to users for their reference through the user's implicit feedback information. The scenario is a sorting task, so a sorting model is needed to model it. In the recommender system, typically, there are two major ways to evaluate the performance of the recommender system, which is based on the two scenarios mentioned above: For rating prediction, RMSE and MAE are used. For Top-N Recommendation, the evaluation metrics are varied: Precision, Recall, F1, Normalized Discounted Cumulative Gain (NDCG), Rank Score, AUC for Receiver Operator Characteristic (ROC) and Catalogue Coverage.

Below is a brief introduction to different algorithms based on Matrix Factorization:

### 3.2.3 Singular Value Decomposition (SVD)

When it comes to matrix factorization, the first thing that comes to mind is the classic SVD in mathematics. The formula is as follows:

$$M_{m\times n} = U_{m\times k}\Sigma_{k\times k}V^T_{k\times n} \tag{2}$$

The form of SVD decomposition is the multiplication of 3 matrices, the left and right matrices respectively represent the user/item implicit factor matrix, the middle matrix is a singular value matrix and a diagonal matrix, and each element satisfies non-negativity and gradually decreases. So we can only need the first k factors to represent it.

$$\overset{\hat{X}}{\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix}_{m\times n}} \approx \overset{U}{\begin{pmatrix} u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \\ u_{m1} & & u_{mr} \end{pmatrix}_{m\times r}} \overset{S}{\begin{pmatrix} s_{11} & 0 & \cdots \\ 0 & \ddots & \\ \vdots & & s_{rr} \end{pmatrix}_{r\times r}} \overset{V^T}{\begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \\ v_{r1} & & u_{rn} \end{pmatrix}_{r\times n}} \tag{3}$$

If we want to use SVD decomposition, the premise is that the matrix is dense. That is, the elements in the matrix must be non-empty. Otherwise, SVD decomposition cannot be used. Obviously, our task cannot use SVD directly. The general method is to first fill the matrix with mean or other statistical methods, and then use SVD decomposition to reduce the dimension.

#### 3.2.4 FunkSVD

As we mentioned before, PureSVD needs to fill the matrix and then decompose and reduce the dimension. At the same time, because of the inverse operation (complexity O ($n^3$)) and the problem of high computational complexity, Simon Funk later proposed The method of FunkSVD is introduced. Instead of decomposing the matrix into three matrices, it decomposes into two low-rank user-item matrices, while reducing the computational complexity:

$$\min_{q^*,p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_T^i p_u)^2 \tag{4}$$

It draws on the idea of linear regression and seeks the optimal implicit vector representation of users and items by minimizing the square of the observed data. At the same time, in order to avoid overfitting observation data, a FunkSVD with L2 regular term is also proposed:

We also have Probabilistic matrix factorization (PMF)[7] as a probabilistic interpretation version of FunkSVD.

#### 3.2.5 BiasSVD

After the introduction of FunkSVD, many variants have been proposed one after another. The relatively popular method is BiasSVD, which is based on the assumption that some users will bring some characteristics, such as being willing to give others praise, being kind-hearted and easy to speak. Some people are demanding. They always score no more than 3 points (On a scale of 1 - 5). At the same time, there are some such projects that determine their status as soon as they are produced. Some are popular, and others are disgusted. This is exactly the reason why the user and project offset items are proposed; The explanation is: for a scoring system, some inherent attributes have nothing to do with the user's items, while some users have some attributes that have nothing to do with the items, and some items have attributes that are not with the user Nothing, the specific prediction formula is as follows:

$$\hat{r}_{ij} = \mu + b_u + bi + q_i^T p_u \tag{5}$$

U is the average rating of the entire website, which is the tone of the entire website; $B_u$ is the user's rating bias, which represents the tone of a user's rating, $B_i$ is the biased rating of an item, which represents the Property tone.[8]

#### 3.2.6 SVD++

In addition to users' explicit ratings, implicit feedback information also helps users to model their preferences, so SVD ++ was subsequently proposed. It is based on the assumption that in addition to the user 's explicit historical scoring records for items, implicit feedback information such as browsing records or favourite lists can also reflect user preferences to a certain extent from the side, such as the user favourited an item It can be reflected from the side that he is interested in this project, and the specific reflection formula is:

$$\hat{r}_{ij} = \mu + b_u + bi + q_i^T (p_u + |N(i)|^{-\frac{1}{2}} \sum_{s\in N(i)} y_s) \tag{6}$$

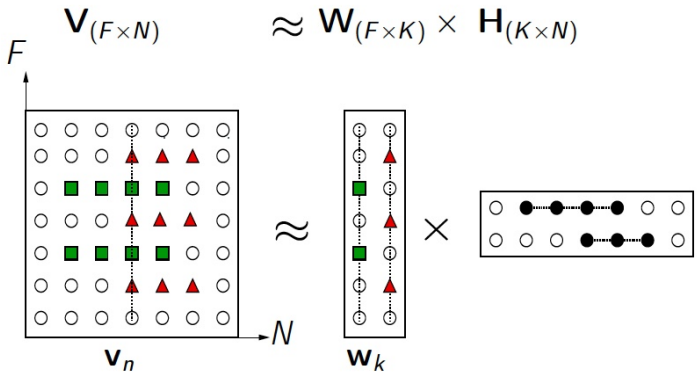
$N(i)$ is the item collection of hidden feedback behaviour generated by the user i; $Y_s$ is a hidden bias of personal preference for the item [s], which is a parameter we want to learn; As for $|N(i)|^{-\frac{1}{2}}$ is an experience formula.[9]

### 3.2.7 timeSVD

It is based on the assumption that the user's interests or substitutions are not static but change dynamically over time. Therefore, timeSVD is proposed, in which the preference between the user and the item will change as time goes, and the user's implicit factor also changes dynamically with time. The implicit representation of the item does not change with time (assuming that the property of the item does not occur in time).[10]

$$\hat{r}_{ui} = \mu + b_i(t_{ui}) + b_u(t_{ui}) + q_i^T p_u(t_{ui}) \tag{7}$$

### 3.2.8 NMF




Because in most methods, the original matrix [formula] is approximately decomposed into two low-rank matrices [formula] multiplied, these methods have in common that even if the elements of the original matrix are non-negative, there is no guarantee The decomposed small matrices are non-negative, which leads to the classic matrix decomposition method in the recommender system can achieve good prediction performance, but can not make a recommendation explanation that is consistent with people's habits like User-based CF. People with similar taste also bought this product). In a mathematical sense, it does not matter if the result of the decomposition is negative. As long as the elements of the matrix after reduction are non-negative and the error is as small as possible, while negative elements are often meaningless in the real world. For example, there can be no negative pixel values in the image data, because the value is between 0 and 255; when counting the word frequency of the document, the negative value cannot be explained. Therefore, the proposed matrix factorization with non-negative constraints is an attempt to scientifically explain the traditional matrix factorization. [11] Its formula is as follows:

$$\begin{aligned} R &\approx P^T Q \\ \text{s.t.} \quad & P \geq 0 \\ & Q \geq 0 \end{aligned} \tag{8}$$

The elements in the two matrices P and Q satisfy non-negative constraints.

### 3.2.9 WMF

For matrix factorization, we generally process the score prediction task in a recommender system, but the same can be used to make Top-N recommendations, that is, to predict whether a user clicks on an item based on implicit information. You can think of it as a binary classification problem, that is, click or not. But this is not an ordinary binary classification problem, because the negative samples are not all true negative samples during the model training process. It may be that the user has not seen the item at all. Maybe he likes it after seeing it. That is, a positive sample tells us what the author likes, but a negative sample does not tell us that the user doesn't like it. Since only positive samples

exist, we define a problem with only positive feedback as a one-class problem, that is, a single-class problem. For single-type problems, there are two solutions, one is the weighted matrix factorization, and the other is the negative sampling technique. Although it was only a bit more weighted, it seemed more naive, but in the context of the research at the time, this small step was actually a big step in the recommender system.[12]

$$L(X) = \sum_{ij} W_{ij}(R_{ij} - Xij)^2 \tag{9}$$

The research on single-type problems has never stopped. Although the negative sampling technique is heuristic, that is, it does not use data modelling to make predictions, the effect is still very useful. In recent years, a model-based approach has been proposed to deal with this kind of single-type problem, that is, modelling from missing data.

#### 3.2.10 LLORMA

The classic matrix factorization model assumes that the entire user-item matrix (that is, the UI matrix) meets the low-rank hypothesis (that is, the global low-rank hypothesis), that is, in the entire system, users and items always meet a certain pattern that is similar, that is, the objects are clustered in classes. People are divided into groups.[13]

This assumption makes sense, but in the current era of big data, the low-rank assumption in the global sense seems too strong, especially in the case of huge amounts of data (that is, in systems with many users and projects). Therefore, this theory overturns the classic global low-rank hypothesis in the global sense. It believes that there are thousands of worlds and that we should find local low-rank hypotheses (that is, local low-rank hypotheses). First, the entire large matrix is divided into several small matrices according to some similarity measure, and each small matrix satisfies a certain similarity threshold, and then a low-rank hypothesis is made in the small local matrix. In this way, the large global matrix can be composed of a plurality of small local matrices in a weighted combination.

#### 3.2.11 SRui

Although the classic matrix factorization method can already achieve relatively good prediction performance, its inherent disadvantages are still inevitable, that is, data sparseness and cold start problems. To alleviate data sparseness, we can introduce rich social information. That is, if two users are friends, then we assume that they have the same preferences, and at the same time, the users they learn implicitly should have a similar distance in the vector space. The user dimension is the same. Similarly, the item dimension can also use this idea to constrain the implicit representation of the item. That is, if the relationship between the two items is close, the distance in the low-dimensional vector space should also be small. The item relationship here is extracted from the UI matrix and becomes the hidden social relationship of the project.[14]

#### 3.2.12 PCA & its variations

Principal Components Analysis is a powerful technique of dimension reduction. This technique uses a transformation that converts possibly correlated observations on certain variables into linearly, not correlated variables, which is called Principal Components. The number of Principal Components equals to or is less than the total number of original variables. This technique is sensitive to how you scale the original observations in each variable.

The Algorithm includes calculating the covariance matrix of the original matrix and then calculating the eigenvectors and the eigenvalues of the matrix, which is the main part of the algorithm and the reason why principal components are orthogonal. After this step is done, we then choose the principal components and form the new feature.

## 3.3 Content-Based Filtering (CBF)

Content-based Filtering Recommender (CBF) is also a widely used recommender algorithm in the industry. Since the collaborative filtering recommender algorithm only recommends products based on the user's rating, there may be a cold start problem. If you can make a more intuitive recommendation based on the characteristics of the item and the user's special preferences, you can solve this cold start Although the CBF algorithm needs additional information that depends on items and user preferences, it does not require too many user ratings or group records, that is, only one user can still complete the recommender function and generate an item recommendation list. The goal of the initial design of the CBF algorithm is to recommend interesting text documents, and the algorithm can also be applied to other recommender fields.[15]

The CBF algorithm consists of three steps: (1)Item Representation: Extract some feature attributes for each item, which is the description operation of the structured item. The corresponding process is called the Content Analyzer. (2)Profile Learning is to use a user's past or dislike item feature data to learn the user's preference profile. (3)Recommender Generation: By comparing the characteristics of the user profile and the featured item obtained in the previous step, the user can recommend a set of items with the highest correlation; the corresponding processing process is called: Filtering Component.

### 3.3.1 CBF-Item Representation

The methods used in machine learning for the extraction of feature characteristics of items mainly include the processing of numerical and non-numerical data. The main processing methods are as follows: (1) normalization of numerical data; (2) binary value of numerical data. (3) Non-numerical data bag-of-words conversion to feature vectors; (4) TF-IDF; (5) Word2Vec.

### 3.3.2 CBF-Profile Learning

We assume that user u has given preference judgments for some items, then the process is to build a discriminative model based on these past judgments of user u. According to this model, we can judge user U will like a new item or not. Therefore, this is a typical supervised learning problem. In theory, machine learning classification algorithms can be used to solve the required discrimination model. Commonly used algorithms are (1) K-Nearest Neighbor; (2) Decision Tree; (3) Linear Classifier; (4) Naive Bayes.

### 3.3.3 Difference between CBF and CF

A content-based recommender system recommends similar items that a given user has liked in the past. It does not require a user-item scoring matrix. The collaborative filtering recommender system will try to identify users with similar interests and recommend items that they have liked. The collaborative filtering recommender system is based on the user-item scoring matrix for the recommendation.

The advantages of CBF are (1) User independence: in the process of building the model, only the current user information need to be considered; (2) Transparency: the content features or descriptions that make the item appear in the recommendation list can be compared Clearly explain how the recommender system works; (3) New items: recommendations can also be made without any scoring.

The disadvantages of CBF are: (1) Limited content that can be analyzed, and feature extraction is more difficult: there are restrictions on the number and types of features related to the recommended object, and it depends on professional knowledge; (2) The potential interest of users cannot be found: due to the recommendation result in CBF is similar to the item that the user previously liked, so if a user only expresses favourite emotions for one item on a website, the recommender system cannot find that the person may also like other items; (3) Cannot generate recommendations for new users: Since the CBF algorithm needs to rely on the user's historical data, it may not be possible to generate a relatively reliable recommendation for new users.

### 3.4 Demographic Filtering (DF)

The demographic-based recommender is one of the easiest recommender algorithms to implement. The algorithm uses only the basic information of the user, such as age and gender, to measure the similarity of users. It recommends items preferred by users with similar tags to users. The process of labelling user information is also commonly referred to as User Profiling. User Profiling is the basic way for an enterprise to apply big data technology after collecting and analyzing data such as consumer social attributes, living habits, and consumer behaviour.[16]

The advantages of DF are: (1)The recommender algorithm only uses the user's basic information data and does not involve the user's historical preference data for items. For new users, as long as we have the basic information of new users, we can make recommendations, and there is no "cold start" problem;(2) The recommender algorithm can be used in different item domains because it does not depend on item information and can be called "domain-independent."

The disadvantages of DF are: (1)The recommender algorithm is currently based on the user's basic information and is relatively rough. In some areas with high taste requirements, such as music and movies, it cannot get a good recommendation effect;(2) The user's basic information is not easy to obtain, such as age; in some cases, the user's basic information also needs to be obtained through modelling.

### 3.5 Knowledge-Based Filtering (KBF)

Traditional recommender algorithms (CBF and CF) are suitable for recommending items with similar characteristics or tastes, such as books, movies, or news. But in the process of recommending certain products, there may not be a particularly suitable method, such as cars, computers, houses, or financial products. There are two main reasons: it is difficult to get a lot of user rating information on a product and users who get recommendations will not produce positive feedback on these outdated products. Knowledge-based Recommendations (KBF) is a new recommender technology that specifically addresses this type of problem. It attaches great importance to knowledge sources and does not have the problem of cold start, because the requirements for recommendations are directly elicited. The disadvantage is that it is difficult to obtain knowledge. Engineers need to organize the knowledge of domain experts into standardized and usable expressions. Knowledge-based recommender technology needs to actively ask the user's needs and then return the recommendation results.[17]

Knowledge-based recommender technology needs to actively ask the user's needs and then return the recommendation results.

The interactive form of conversational recommender system (1) The user specifies his initial preferences, and then collects the answers to all questions at one time or step by step; (2) When enough information about the user's needs and preferences is collected, the user will be provided with a set of matching products; Users may modify their needs; (4) It is similar to the search process, except that the parameters given in the search process are input into a knowledge-based recommender system.

Knowledge-based recommender systems fall into two broad categories: sample-based recommenders and constraint-based recommenders; these two methods are very similar: they both collect user needs first. In the case where no recommended solution can be found, they will automatically fix the inconsistencies with the requirements and give a recommended explanation. The difference lies in how the recommended scheme is calculated. A sample-based recommendation method retrieves items from a catalogue through similarity. Constraint-based recommendation methods mainly use a predefined recommendation knowledge base, such as some display association rules that describe user needs and product information characteristics related to these needs.

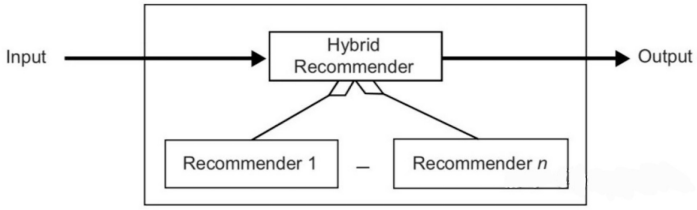
### 3.6 Hybrid Recommender System

We can think of the recommender system as a black box. The input data can be converted into an ordered list of items and then output. The input data mainly includes user records, context

information, product features, knowledge models, and so on. But no recommender algorithm can make use of all the input data, so scholars consider mixing the results of multiple recommender system models as the final recommendation result. The design structure of the hybrid recommender system is mainly divided into three categories, namely: Monolithic hybrid design, parallel hybrid design, and pipeline hybrid design. Each category has two to three specific implementation schemes, with a total of 7 different mixed schemes. We will introduce them separately below.[18]

### 3.6.1 Monolithic hybrid design

The single hybrid paradigm integrates multiple recommender algorithms into the same algorithm system. The integrated recommender algorithm provides recommendation services in a unified manner. The specific implementation process is shown in the figure below.



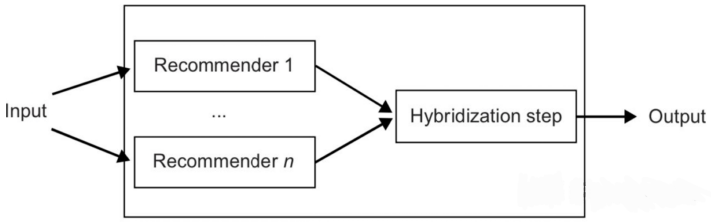

There are two specific implementation schemes:

(1) Feature Combination
Feature combination uses the feature data of multiple recommender algorithms as the original input and uses one of the algorithms as the main algorithm to finally generate the recommendation results. For example, for collaborative filtering and content-based recommendation, we can use collaborative filtering algorithms to assign a feature to each sample, and then use these features and content-related features to build a content-based recommender algorithm based on content recommendation. The combination of collaborative filtering and content-based recommendation for feature combination can enable the recommender system to utilize collaborative data, thus reducing the system's sensitivity to the number of users who have an action on an object. In other words, even if an object does not have much user behaviour, it can still be recommended well. Because the feature combination method is very simple, combining collaborative filtering and content-based recommendation is a very common solution.

(2) Feature Augmentation Blend
Feature enhancement takes advantage of more complex processing and transformations. The first algorithm pre-processes and generates intermediate available features or data, which is then used by the second algorithm to finally generate recommendation results.

### 3.6.2 Parallel hybrid design

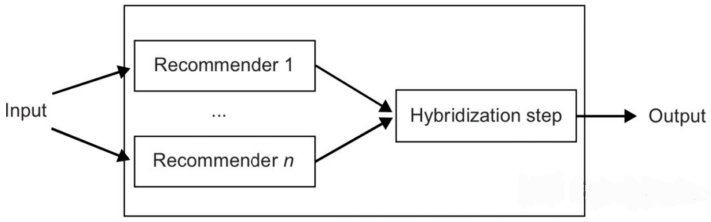
The parallel hybrid design uses multiple independent recommender algorithms. Each recommender algorithm generates its own recommendation results. These recommendation results are fused in the mixing stage to generate the final recommendation results. For the specific implementation logic, refer to the following figure.




There are three specific implementations as follows. (1) The mixed-method unifies the scores predicted by different algorithms to a comparable range. For example, you can first normalize the score of each algorithm to between 0-1. It is then sorted according to the normalized score size, or it can be sorted separately by another algorithm. (2)The weighting method uses the recommendation results of multiple recommender algorithms, obtains the weighted score of each recommended candidate target by weighting, and finally, sorts. It is also necessary to ensure that the scores of different recommender algorithms are in the same range. Otherwise, the weighting is meaningless. (3)The switching method decides which recommender algorithm to use when a certain situation occurs. The branching conditions can be related to user status or context. For example, if the user uses the product in the morning, we recommend news to the user. At this time, the branch condition is time.

### 3.6.3 Pipelined hybrid design

In pipeline hybrid design, the recommendation results generated by the previous recommender algorithm are used as the input of the latter recommender algorithm, and so on. The hybrid logic of it is shown in the figure below.




Pipeline hybrid design is a phased process, which can be specifically divided into the following two implementation schemes. (1)Cascade hybrid design. In the cascade hybrid design, the recommendation result of one algorithm is output to the next algorithm as one of the inputs. The next algorithm will adjust the ranking of the recommendation results of the previous algorithm or remove some of the results. Other data may also be added to use. The purpose of cascading is to optimize the ranking results of the previous algorithm or to eliminate inappropriate recommendations. Through cascading, the number of final recommendation results will be reduced. (2)The meta-level hybrid design directly takes the model as the input of another algorithm. This method is complicated. Generally, only two layers are mixed in a real business scenario.

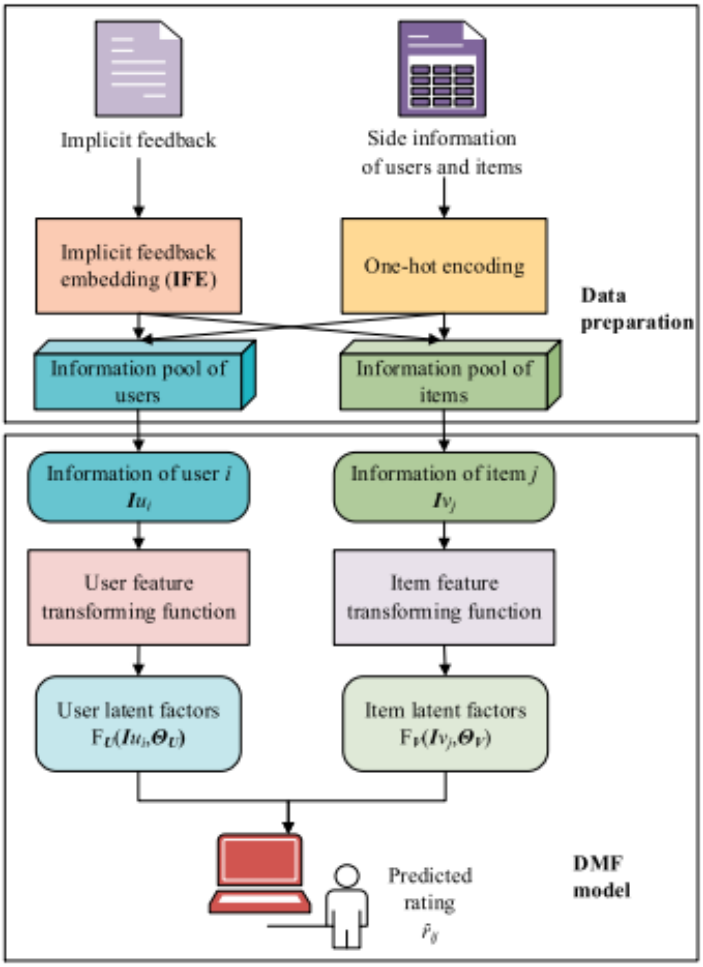
If we don't have any more information about the features, the monolithic hybrid design is valuable, and it only needs to perform little preprocessing and fine-tuning on the main recommender algorithm and data structure. The parallel hybrid design is a way to minimize business intrusion because the mixing phase is simply a mixture of the results of different algorithms. But because of this, the computational complexity of the entire recommendation is higher. And how the scores of the recommendation results of multiple algorithms are comparable in the same framework is also a tricky problem that needs to be handled well. The pipelined hybrid design is the most complex and time-consuming type of hybrid solution. It requires a good understanding of the two algorithms, and they also need to cooperate well to finally produce better results than a single algorithm.

## 3.7 Deep Learning

As S. Zhang et al. mentioned(2019)[19] ,in terms of deep learning models, based on the publications, Deep Learning based recommender systems can be classified into two different categories: Recommender systems with neural blocks and deep hybrid models.

Since there is no brief and systematic introduction for those models, and most deep learning models are based on three base models, here we only give a brief introduction on the three base techniques for most deep learning models:

### 3.7.1 Deep Matrix Factorization




The DMF model can work well with the accuracy of users and items information, which is provided by the data preparation process. In the first process *datapreparationprocess*, the implicit feedback is encoded by IFE, and side information is encoded by one-hot encoding. Combining these two information, users and items information pools could be generated. In the DMF model, the predicted rating could be estimated by three steps. First, the target user and item information are extracted from users and items information pools. Second, latent factors are generated from user and item feature transforming functions. Third, predicted rating could be estimated by latent factors.[20]

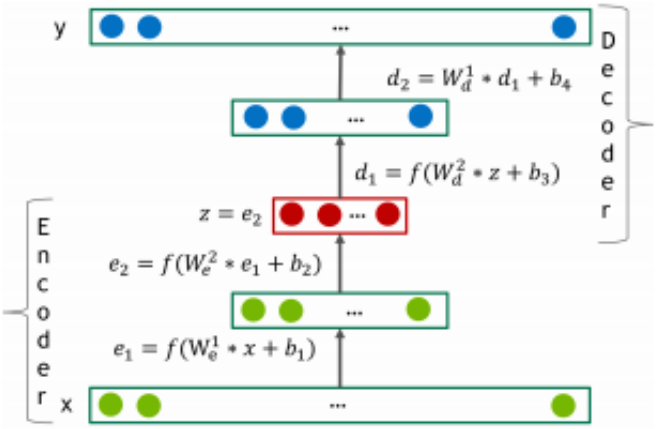
### 3.7.2 Restricted Boltzmann Machines

As the name suggests, Restricted Boltzmann Machines are a variant of Boltzmann Machine. It's widely used in dimension reduction, classification and recommender system, the training itself can be either supervised or unsupervised learning. The basic structure including two layers: visible layer and hidden layer. All neurons in the same layer are not connected and are only connected with neurons in the other layers, which is why it's called restricted, as no two nodes in the same layers are connected. [21]

RBM has two stages: Forward and Backward. It takes the input and transfers them to the output and then that information is transferred back to the input. In this way, a well trained RBM can tell us which features are the most important among all the features.

### 3.7.3 Autoencoder

An autoencoder is a network that consists of two different transformations: encoder and decoder. It is an unsupervised learning model. The model summarizes or compresses the input from a multidimensional data into a low dimensional code and then rebuilds the output from this representation. The code here is a summary or compressed version of the input; it is also called latent-space representation or latent variables.[22]




The figure shown is an example of an autoencoder with two encoder layers and two decoder layers. Autoencoder is a good technique for de-nosing, as it would drop out noise nodes in both the input layer and code layer. The whole architecture is similar to a feed-forward neural network.

# 4 R Resources Review

We have found several libraries in R that can be applied to this problem. They are recommenderlab, rrecsys, recosystem, rectool and NMF. In general, recommenderlab and recosystem are the most popular libraries in this area on R, as they can be seen in many different recommender system introductions, tutorials or comparisons. However, recosystem only has MF available to its user, while recommenderlab and rrecsys have more algorithms and evaluation methods available. As for rectools, it's not a CRAN library, but it provides some unique features like focus finder and plot for the model. It is worth mentioning that not a single library provides a complete solution for recommender systems on R and also combine all of them together, there are still a lot of algorithms missing in this field.

## 4.1 recommenderlab

Github Link:https://github.com/mhahsler/recommenderlab
CRAN Website: https://cran.r-project.org/web/packages/recommenderlab/index.html
This library can handle both ratings (1-5) or unary (0 or 1) input dataset and also provide support to several algorithms including[23]:

- Collaborative Filtering
  - User-based (UBCF)
  - Item-based (IBCF)
- Singular Value Decomposition (SVD)
  - SVD with column-mean imputation
  - Funk SVD
- Alternating Least Squares
- Matrix factorization with LIBMF
- Association rule-based recommender
- Popular items
- Randomly chosen items for comparison
- Re-recommend liked items
- Hybrid recommendations

Also this library also support three different data splitting methods including:

- Train-Test Split
- Cross-Validation
- Repeated Bootstrap sampling

And also two ways of validation to choose from:

- If predicting Rating: MSE, RMSE, MAE
- If predicting Top-N Recommendation: TPR (True Positive Rate), FPR (False Positive Rate) (ROC curve), Precision and Recall.

This library provided a from beginning to end throughout solution to our problem, this library would be used in the next chapter for example.

## 4.2 rrecsys

Github Link:https://github.com/ludovikcoba/rrecsys
CRAN Website: https://cran.r-project.org/web/packages/rrecsys/index.html
This library provides some implementation of algorithms in the recommender system. The algorithm it provides are mainly in two different categories[24]:

- Non-Personalized
  - Global Average

  - Item Average
  - User Average
  - Most Popular
- Collaborative filtering
  - Item-bsaed KNN
  - Funk SVD
  - Bayesian Personalized Ranking
  - Weighted Alternated Least Squares

All but Bayesian Personalized Ranking and Weighted Alternated Least Squares support both scale rating and binary rating, the latter two methods only support One-Class. This library is also presented in the next chapter

For the evaluation:

- Rating: Global or Item-based MAE and RMSE
- Recommendation: Precision, Recall, F1, NDCG, rank score and confusion Matrix

## 4.3 recosystem

Github Link:https://github.com/yixuan/recosystem

CRAN Website: https://cran.r-project.org/web/packages/recosystem/index.html

This library is a R wrapper of 'libmf' library which is a Matrix Factorization library. It provides train, tune and prediction of the Matrix Factorization recommender system. However, the evaluation and split part should be done by the user. One advantage of this library is that it can save a lot of memory compared with all other R libraries as it wouldn't save the whole dataset in the memory.[25]

## 4.4 rectools

Github Link:https://github.com/Pooja-Rajkumar/rectools

CRAN Website: N/A

Rectools is also called Advanced Package for Recommender Systems. It provides a lot of algorithms including:

- NMF
- ANOVA
- cosine
- Hybrid system

Also it has some unique functions including:

- Plot
- Focus Group Finder
- Parallel Computation
- Enhancement of existing libraries

### 4.5 NMF

Github Link: https://github.com/renozao/NMF
CRAN Website: https://cran.r-project.org/web/packages/NMF/index.html
For Non-negative Matrix Factorization and variants, we can use library NMF, rNMF, NNLM and fastFM. Due to space limitations, we will only introduce NMF as follows. This library has several built-in algorithms including:
This library has several built-in algorithms including[26]:

- Standard NMF
  - Based on Kullback-Leibler divergence
  - Based on euclidean distance
- NMF with an intercept
- Non-smooth NMF
- Pattern-Expression NMF
- Alternating Least Square

And there are 4 seeding methods provided:

- ICA (Independent Component Analysis
- Non-negative Double Singular Value Decomposition.
- Fix seed
- Random seed that are drawn from a uniform distribution

Additionally this package support custom algorithms and seeding methods.

## 5 Instances of Implementation

In this section, a tutorial on recommender system using *recommenderlab* and *rrecsys* package will be given. This tutorial also shows the implementation of several different algorithms and provides a comparison among those algorithms and models.

### 5.1 Dataset

The dataset used in this section is MovieLens 1M dataset from https://movielens.org. MovieLens is one of the widely used datasets in the research field of the recommender system. The Department of Computer Science and Engineering at the University of Minnesota established GroupLens Research in 1992, which provides MovieLens data in different sizes.
The dataset we are using has 1,000,209 observations, 6,040 users and 3,706 movies and with only integer ratings on the scale of 1 - 5 and was released on Feb. 2003.

### 5.2 Preparation

Before going into deep, we should install and import libraries needed for this example and read the data from the CSV file:
**NOTICE: When installing package *SVDApproximation* from Github, if your console in RStudio informs you to update specific packages, please input 1 and press "Enter" button on your keyboard.**

### 4.5 NMF

Github Link: https://github.com/renozao/NMF
CRAN Website: https://cran.r-project.org/web/packages/NMF/index.html
For Non-negative Matrix Factorization and variants, we can use library NMF, rNMF, NNLM and fastFM. Due to space limitations, we will only introduce NMF as follows. This library has several built-in algorithms including:
This library has several built-in algorithms including[26]:

- Standard NMF
  - Based on Kullback-Leibler divergence
  - Based on euclidean distance
- NMF with an intercept
- Non-smooth NMF
- Pattern-Expression NMF
- Alternating Least Square

And there are 4 seeding methods provided:

- ICA (Independent Component Analysis
- Non-negative Double Singular Value Decomposition.
- Fix seed
- Random seed that are drawn from a uniform distribution

Additionally this package support custom algorithms and seeding methods.

## 5 Instances of Implementation

In this section, a tutorial on recommender system using *recommenderlab* and *rrecsys* package will be given. This tutorial also shows the implementation of several different algorithms and provides a comparison among those algorithms and models.

### 5.1 Dataset

The dataset used in this section is MovieLens 1M dataset from https://movielens.org. MovieLens is one of the widely used datasets in the research field of the recommender system. The Department of Computer Science and Engineering at the University of Minnesota established GroupLens Research in 1992, which provides MovieLens data in different sizes.
The dataset we are using has 1,000,209 observations, 6,040 users and 3,706 movies and with only integer ratings on the scale of 1 - 5 and was released on Feb. 2003.

### 5.2 Preparation

Before going into deep, we should install and import libraries needed for this example and read the data from the CSV file:
**NOTICE: When installing package *SVDApproximation* from Github, if your console in RStudio informs you to update specific packages, please input 1 and press "Enter" button on your keyboard.**

```
options(repos = list(CRAN="http://cran.rstudio.com/"))
install.packages("recommenderlab",quiet=TRUE)
install.packages("rrecsys",quiet=TRUE)
library(devtools,quietly = TRUE)
devtools::install_github("tarashnot/SVDApproximation",quiet =TRUE)
library(recommenderlab,quietly = TRUE)

##
## Attaching package:  'arules'
## The following objects are masked from 'package:base':
##
##    abbreviate, write
##
## Attaching package:  'proxy'
## The following object is masked from 'package:Matrix':
##
##    as.matrix
## The following objects are masked from 'package:stats':
##
##    as.dist, dist
## The following object is masked from 'package:base':
##
##    as.matrix

library(SVDApproximation,quietly = TRUE)

## Warning:  replacing previous import 'data.table::melt' by 'reshape2::melt' when loading
'SVDApproximation'
## Warning:  replacing previous import 'data.table::dcast' by 'reshape2::dcast' when loading
'SVDApproximation'

library(data.table,quietly = TRUE)
library(RColorBrewer,quietly = TRUE)
library(ggplot2,quietly = TRUE)

ratings <- read.csv("/Users/miaohu/Downloads/Miao_Hu_Guangmei_Ye/Data.csv")
```

Although we encountered a warning here, it is caused by conflicts of functions with the same name but in different packages. It wouldn't cause a problem in the following implementation.

After the data were read by R, it should be converted to a sparse format, which we can use sparseMatrix function to approach that easily.

```
sparse_ratings <- sparseMatrix(i = ratings$user, j = ratings$item, x = ratings$rating,
                               dims = c(length(unique(ratings$user)),
                                        length(unique(ratings$item))),
                               dimnames = list(paste("u", 1:length(unique(ratings$user)),
                                                     sep = ""),
                              paste("m",
```

```
                                    1:length(unique(ratings$item)),sep = ""))
```

The sparseMatrix format of our data would look like:

```
sparse_ratings[1:10, 1:10]

## 10 x 10 sparse Matrix of class "dgCMatrix"

##    [[ suppressing 10 column names 'm1', 'm2', 'm3' ...  ]]

##
## u1  5 . . . . . . . . .
## u2  . . . . . . . . . .
## u3  . . . . . . . . . .
## u4  . . . . . . . . . .
## u5  . . . . . 2 . . . .
## u6  4 . . . . . . . . .
## u7  . . . . . 4 . . . .
## u8  4 . . 3 . . . . . .
## u9  5 . . . . . . . . .
## u10 5 5 . . . . 4 . . .
```

Finally we convert it to *realRatingMatrix* class that could be processed by *recommenderlab*

```
real_ratings <- new("realRatingMatrix", data = sparse_ratings)
real_ratings

## 6040 x 3706 rating matrix of class 'realRatingMatrix' with 1000209 ratings.
```

## 5.3 Exploratory Data Analysis

First of all, we should do some Exploratory Data Analysis to get to know our data better.
As we can see, this dataset contains User ID, Item ID and ratings.

```
print(head(ratings,10))

##     user item rating
##  1:    1    1      5
##  2:    6    1      4
##  3:    8    1      4
##  4:    9    1      5
##  5:   10    1      5
##  6:   18    1      4
##  7:   19    1      5
##  8:   21    1      3
##  9:   23    1      4
## 10:   26    1      3
```

Here we can see some summary statistics of *ratings*:

In the figure above, we can see that the User's average ratings are 3.62 and items' average ratings is 3.16 while the most frequent rating is 4.

Also, we can see the Median of rated items by User is 65 while the Median of ratings an item has is 27.

```
visualize_ratings(ratings_table = ratings)
```


## 5.4 Recommenders

In this section, we will apply multiple recommender methods within the *recommenderlab* and *rrecsys* libraries, including *Popular, User-Based Collaborative Filtering, Item-Based Collaborative Filtering, SVD, FunkSVD and user-based KNN.*

### 5.4.1 Popular

Our benchmark is to use the item average(also named most popular). It computes the mean rating for each item based on available ratings and predicts each unknown based on the average rating. Thus, missed ratings for each item will be the same. Like we mentioned before in centered cosine, We normalized the data by subtracting means of each User from all their ratings to avoid bias.

Here is an example of the first 5 users and 5 movies.

```
model <- Recommender(real_ratings, method = "POPULAR", param=list(normalize = "center"))
prediction <- predict(model, real_ratings[1:5], type="ratings")
as(prediction, "matrix")[,1:5]

##          m1       m2       m3       m4       m5
## u1       NA 3.864537 3.748432 3.489864 3.768903
## u2 4.192675 3.389036 3.272931 3.014363 3.293402
## u3 4.381458 3.577818 3.461714 3.203145 3.482184
## u4 4.669973 3.866334 3.750229 3.491661 3.770700
## u5 3.625962 2.822322 2.706218 2.447649 2.726688
```

Also we split the data into 80% train and 20% validation dataset for evaluation.

```
e <- evaluationScheme(real_ratings, method="split", train=0.8, given=-5)
```

Now, let's apply Item Average and get the RMSE and computation time for the predictions:

```
set.seed(1)
time1 <- Sys.time()

model <- Recommender(getData(e, "train"), "POPULAR")
prediction <- predict(model, getData(e, "known"), type="ratings")
time2 <- Sys.time()

rmse_popular <- calcPredictionAccuracy(prediction, getData(e, "unknown"))[1]
print(time2-time1)

## Time difference of 2.27263 secs

rmse_popular

##      RMSE
## 0.9797912
```

### 5.4.2 User-Based Collaborative Filtering

As we mentioned before, User-based CF generates the results depending on the ratings from its closest users. The nn here is a hyperparameter that we can tune to get better performance. For example, we set nn = 100 to find out the top 100th nearest neighbours and use the centered cosine method to calculate the similarity.

Here shows the small sample (five users and five movies) for prediction:

```
model <- Recommender(real_ratings, method = "UBCF",
                     param=list(normalize = "center", method="Cosine", nn=100))

#Making predictions
prediction <- predict(model, real_ratings[1:5], type="ratings")
as(prediction, "matrix")[,1:5]

##          m1       m2       m3       m4       m5
```

```
## u1       NA 4.188679 4.122541 4.188679 4.188679
## u2 3.807039 3.761195 3.722409 3.713178 3.695678
## u3 3.901223 3.909234 3.888925 3.893896 3.901961
## u4 4.308763 4.205048 4.138600 4.190476 4.198906
## u5 3.183357 3.151116 3.157666 3.146465 3.135898
```

Now we use the recommender function to train the model and estimate the RMSE for the prediction.

```
set.seed(1)
time3 <- Sys.time()

model <- Recommender(getData(e, "train"), method = "UBCF",
                     param=list(normalize = "center", method="Cosine", nn=100))

prediction <- predict(model, getData(e, "known"), type="ratings")

time4 <- Sys.time()

rmse_ubcf <- calcPredictionAccuracy(prediction, getData(e, "unknown"))[1]

print(time4-time3)

## Time difference of 2.148755 mins

rmse_ubcf

##    RMSE
## 1.05187
```

### 5.4.3 Item-Based Collaborative Filtering

Item-Based CF has the same principle as User-Based; the difference is that the item-based is trying to capture similar items. We set k=50 to calculate the similarity based on the score from the top 50th nearest items. In general, we can try different parameter settings through cross-validation to find the optimal parameters. Due to space limitations, we have omitted to show the process of adjusting parameters. We believe that after understanding the principles and algorithms, tuning parameters is easy to implement.

Here is the small sample (five users and five movies) for prediction:

```
model <- Recommender(real_ratings, method = "IBCF",
                     param=list(normalize = "center", method="Cosine", k=50))
#Making predictions
prediction <- predict(model, real_ratings[1:5], type="ratings")
as(prediction, "matrix")[,1:5]

##    m1 m2 m3 m4 m5
## u1 NA NA NA NA NA
## u2 NA NA NA NA NA
## u3 NA NA NA NA NA
## u4 NA NA NA NA NA
## u5 NA NA NA NA NA
```

And now, we use recommender function to train the model and estimate the RMSE for the prediction.

```
set.seed(1)
time5 <- Sys.time()

model <- Recommender(getData(e, "train"), method = "IBCF",
                     param=list(normalize = "center", method="Cosine", k=50))

prediction <- predict(model, getData(e, "known"), type="ratings")
time6 <- Sys.time()

rmse_ibcf <- calcPredictionAccuracy(prediction, getData(e, "unknown"))[1]

print(time6-time5)

## Time difference of 6.580528 mins

rmse_ibcf

##     RMSE
## 1.700758
```

### 5.4.4 Singular Value Decomposition

In SVD, the Rating Matrix is decomposed and then reconstructed to keep only the first k rows. Here k also means the number of latent factors of decomposition. We can find the best value for this hyperparameter through cross-validation. It is worth mentioning that a higher k would cause overfitting.

Firstly, we'll take a look at the prediction examples on the first 5 users and movies.

```
model <- Recommender(real_ratings, method = "SVD",
                     param=list(normalize = "center", k=50))
#Making predictions
prediction <- predict(model, real_ratings[1:5], type="ratings")
as(prediction, "matrix")[,1:5]

##          m1       m2       m3       m4       m5
## u1       NA 4.200229 4.184380 4.190234 4.192867
## u2 3.873968 3.628721 3.774081 3.749153 3.731536
## u3 3.983118 3.874014 3.979973 3.914418 3.924176
## u4 4.260126 4.183906 4.254089 4.199484 4.235967
## u5 3.144652 3.101921 3.311447 3.138256 3.179611
```

Then, we shall predict using the complete dataset and evaluate through RMSE and computation time.

```
time7 <- Sys.time()

model <- Recommender(getData(e, "train"), method = "SVD",
                     param=list(normalize = "center", k=50))
```

```
prediction <- predict(model, getData(e, "known"), type="ratings")
time8 <- Sys.time()

rmse_svd <- calcPredictionAccuracy(prediction, getData(e, "unknown"))[1]

print(time8-time7)

## Time difference of 12.19937 secs

rmse_svd

##     RMSE
## 1.012826
```

### 5.4.5 FunkSVD

FunkSVD is similar to PureSVD. However, PureSVD needs to fill the matrix, but FunkSVD can work on a sparse matrix.

We use evalPred from library rrecsys to predict using the test dataset and evaluate through RMSE and computation time.

First of all, we need to detach the recommenderlab library since these two packages have some conflicts.

```
detach(package:recommenderlab)
library(rrecsys,quietly = TRUE)

##
## Attaching package:  'rrecsys'
## The following object is masked from 'package:arules':
##
##    predict
## The following object is masked from 'package:stats':
##
##    predict
```

Then we convert the data from *realRatingMatrix* to *matrix*, as the former format is not accepted by *rrecsys* package.

```
set.seed(1)

ratings_matrix <- as(real_ratings,"matrix")

d <- defineData(ratings_matrix)
e <- evalModel(d, folds = 5)
mf_model <- evalPred(e, "funk", k = 10, steps = 100, regCoef = 0.0001,
                     learningRate = 0.001, biases = F)

## Train set 1 / 5 created with  6040 users,  3706 items and  800138  ratings.
## Features trained. Time: 11.11044 seconds.
##
## Fold: 1 / 5  elapsed. Time: 17.67627
##
```

```
## Train set 2 / 5 created with  6040 users,  3706 items and  800183  ratings.
## Features trained. Time: 11.13879 seconds.
##
## Fold: 2 / 5  elapsed. Time: 17.82178
##
## Train set 3 / 5 created with  6040 users,  3706 items and  800192  ratings.
## Features trained. Time: 11.16819 seconds.
##
## Fold: 3 / 5  elapsed. Time: 17.78521
##
## Train set 4 / 5 created with  6040 users,  3706 items and  800176  ratings.
## Features trained. Time: 11.30802 seconds.
##
## Fold: 4 / 5  elapsed. Time: 17.90596
##
## Train set 5 / 5 created with  6040 users,  3706 items and  800147  ratings.
## Features trained. Time: 11.23841 seconds.
##
## Fold: 5 / 5  elapsed. Time: 17.89855
##
## The model was trained on the dataset using  FunkSVD algorithm.
## The algorithm was configured with the following parameters:
##    k learningRate regCoef biases
## 1 10        0.001   1e-04  FALSE

rmse_funkSVD <- mf_model$RMSE[6]
timedif1 <- mf_model$Time[6]
rmse_funkSVD

## [1] 0.8693407
```

### 5.4.6 User-Based K-Nearest Neighbors

This algorithm identifies the k-nearest neighbours (or we can say the most similar users) of a given node.

There are two algorithms to choose from to measure the similarity: Cosine and Pearson Correlation, here we choose to present the Pearson Correlation.

```
set.seed(1)

d <- defineData(ratings_matrix)
e <- evalModel(d, folds = 5)
mf_model <- evalPred(e, "ubknn", simFunct = "Pearson", neigh = 5)

## Train set 1 / 5 created with  6040 users,  3706 items and  800138  ratings.
## Neighborhood calculated in:  444.2002 seconds.
##
## Fold: 1 / 5  elapsed. Time: 451.6937
##
## Train set 2 / 5 created with  6040 users,  3706 items and  800183  ratings.
## Neighborhood calculated in:  435.7676 seconds.
```

```
##
## Fold: 2 / 5  elapsed. Time: 443.0012
##
## Train set 3 / 5 created with  6040 users,  3706 items and  800192  ratings.
## Neighborhood calculated in:  435.4433 seconds.
##
## Fold: 3 / 5  elapsed. Time: 442.8779
##
## Train set 4 / 5 created with  6040 users,  3706 items and  800176  ratings.
## Neighborhood calculated in:  434.9788 seconds.
##
## Fold: 4 / 5  elapsed. Time: 442.2041
##
## Train set 5 / 5 created with  6040 users,  3706 items and  800147  ratings.
## Neighborhood calculated in:  438.7772 seconds.
##
## Fold: 5 / 5  elapsed. Time: 445.8384
##
## The model was trained on the dataset using  UBKNN algorithm.
## The algorithm was configured with the following neighborhood width: 5

rmse_ubknn <- mf_model$RMSE[6]
timedif2 <- mf_model$Time[6]
rmse_ubknn

## [1] 0.9954409
```

### 5.4.7 Comparison

In conclusion, we have implemented 5 different algorithms from the same package, let's compare them in terms of computation expense and RMSE.

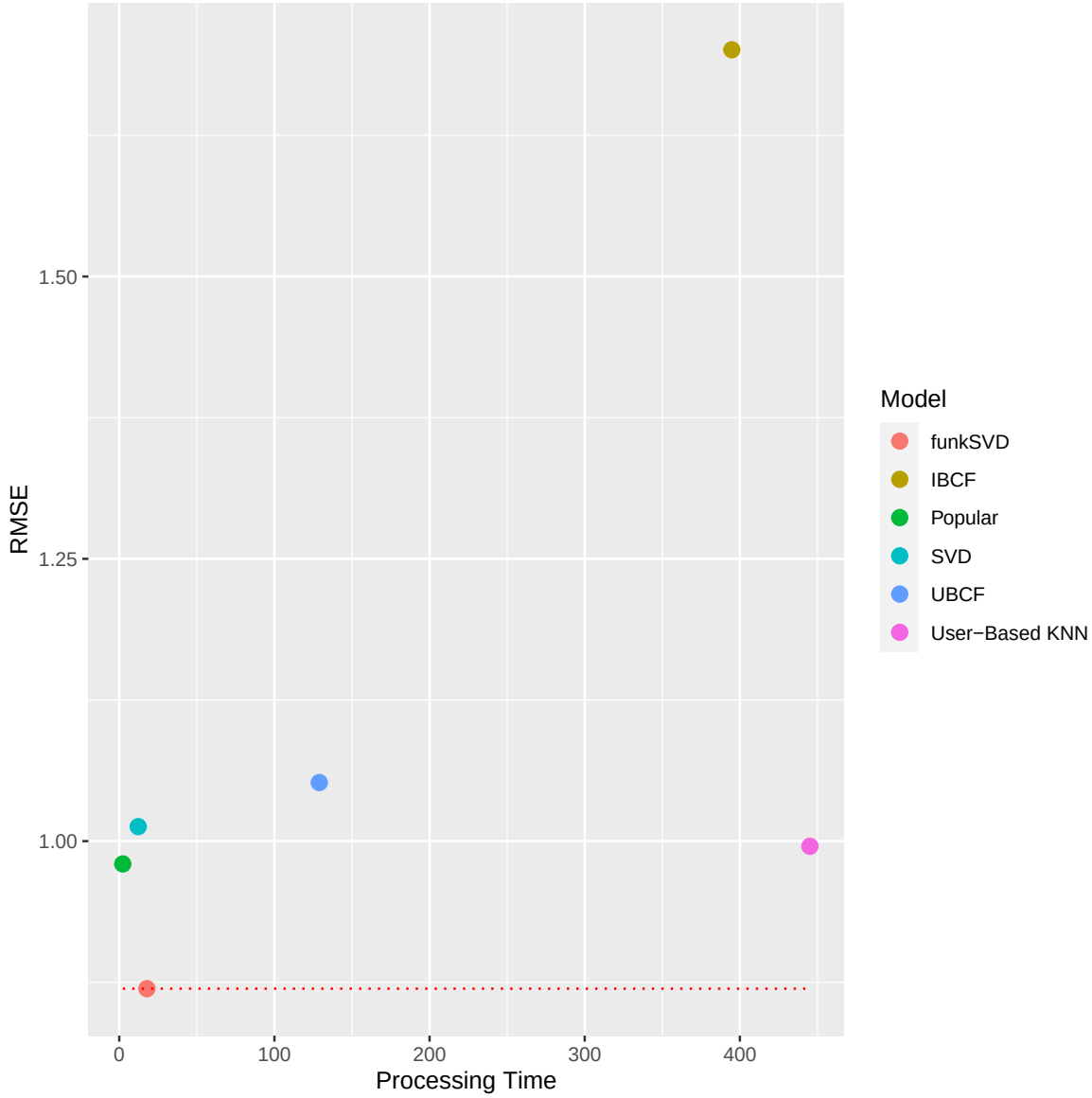
```
modelname <- c("Popular","UBCF","IBCF","SVD", "funkSVD","User-Based KNN")
time <- c(time2-time1,time4-time3,time6-time5,time8-time7,timedif1,timedif2)
RMSE <- c(rmse_popular,rmse_ubcf,rmse_ibcf,rmse_svd,rmse_funkSVD,rmse_ubknn)

comparison <- data.frame(Model = modelname,ProcessingTime = time,RMSE = RMSE)
print(comparison)

##            Model ProcessingTime      RMSE
## 1        Popular   2.27263 secs 0.9797912
## 2           UBCF 128.92529 secs 1.0518701
## 3           IBCF 394.83166 secs 1.7007579
## 4            SVD  12.19937 secs 1.0128260
## 5        funkSVD  17.81745 secs 0.8693407
## 6 User-Based KNN 445.12295 secs 0.9954409
```

In the above table and graph below, we can see that although Popular is the best methods on computation time, FunkSVD is the best among all methods, with a little cost of computation time it reached a lower RMSE.

```r
ggplot(comparison, aes(as.numeric(ProcessingTime), RMSE, colour = Model)) +
  geom_point(size=3)+
  xlab("Processing Time")+
  geom_line(aes(y = min(RMSE)), color = "red", linetype = "dotted")
```




# References


[1] Erion Çano and Maurizio Morisio. "Hybrid recommender systems: A systematic literature review". In: *Intelligent Data Analysis* 21.6 (2017), pp. 1487–1524.

[2] Joeran Beel et al. "paper recommender systems: a literature survey". In: *International Journal on Digital Libraries* 17.4 (2016), pp. 305–338.

[3] Ken Goldberg et al. "Eigentaste: A constant time collaborative filtering algorithm". In: *information retrieval* 4.2 (2001), pp. 133–151.

[4] Bin Wang, Qing Liao, and Chunhong Zhang. "Weight based KNN recommender system". In: *2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics*. Vol. 2. IEEE. 2013, pp. 449–452.

[5] Anil Poriya et al. "Non-personalized recommender systems and user-based collaborative recommender systems". In: *Int. J. Appl. Inf. Syst* 6.9 (2014), pp. 22–27.

[6] Gleb Beliakov, Tomasa Calvo, and Simon James. "Aggregation of preferences in recommender systems". In: *Recommender systems handbook*. Springer, 2011, pp. 705–734.

[7] Andriy Mnih and Russ R Salakhutdinov. "Probabilistic matrix factorization". In: *Advances in neural information processing systems*. 2008, pp. 1257–1264.

[8] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems". In: *Computer* 42.8 (2009), pp. 30–37.

[9] Yehuda Koren. "Factor in the neighbors: Scalable and accurate collaborative filtering". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 4.1 (2010), pp. 1–24.

[10] Koren Y Collaborative. "filtering with temporal dynamics [J]". In: *Communications of the ACM* 53.4 (2010), pp. 89–97.

[11] Daniel D Lee and H Sebastian Seung. "Learning the parts of objects by non-negative matrix factorization". In: *Nature* 401.6755 (1999), pp. 788–791.

[12] Rong Pan et al. "One-class collaborative filtering". In: *2008 Eighth IEEE International Conference on Data Mining*. IEEE. 2008, pp. 502–511.

[13] Mao-Lin Li et al. "Matrix Factorization with Interval-Valued Data". In: *IEEE Transactions on Knowledge and Data Engineering* (2019).

[14] Hao Ma. "An experimental study on implicit social recommendation". In: *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. 2013, pp. 73–82.

[15] Jieun Son and Seoung Bum Kim. "Content-based filtering for recommendation systems using multiattribute networks". In: *Expert Systems with Applications* 89 (2017), pp. 404–412.

[16] Michael J Pazzani. "A framework for collaborative, content-based and demographic filtering". In: *Artificial intelligence review* 13.5-6 (1999), pp. 393–408.

[17] Robin Burke. "Knowledge-based recommender systems". In: *Encyclopedia of library and information systems* 69.Supplement 32 (2000), pp. 175–186.

[18] Robin Burke. "Hybrid recommender systems: Survey and experiments". In: *User modeling and user-adapted interaction* 12.4 (2002), pp. 331–370.

[19] Shuai Zhang et al. "Deep learning based recommender system: A survey and new perspectives". In: *ACM Computing Surveys (CSUR)* 52.1 (2019), pp. 1–38.

[20] Baolin Yi et al. "Deep matrix factorization with implicit feedback embedding for recommendation system". In: *IEEE Transactions on Industrial Informatics* 15.8 (2019), pp. 4591–4601.

[21] Fan Yang and Ying Lu. "Restricted Boltzmann machines for recommender systems with implicit feedback". In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE. 2018, pp. 4109–4113.

[22] Oleksii Kuchaiev and Boris Ginsburg. "Training deep autoencoders for collaborative filtering". In: *arXiv preprint arXiv:1708.01715* (2017).

[23] Michael Hahsler, Bregt Vereet, and Maintainer Michael Hahsler. *Package 'recommenderlab'*. 2019.

[24] Ludovik Çoba and Markus Zanker. "rrecsys: an R-package for prototyping recommendation algorithms". In: CEUR-WS, 2016.

[25] Yixuan Qiu et al. "Package 'recosystem'". In: (2017).

[26] Renaud Gaujoux and Cathal Seoighe. “A flexible R package for nonnegative matrix factorization”. In: *BMC bioinformatics* 11.1 (2010), p. 367.