PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
IBN KHALDOUN UNIVERSITY – TIARET

MATHEMATICS AND COMPUTER FACULTY
COMPUTER SCIENCE DEPARTMENT

Theme

The Sparse Matrix and Collaborative filtering system

For obtaining Master’s degree

Specialty: Software Engineering

By: LENOUAR Miloud

The jury is composed of:

Mr. TALBI Omar	MCB	Tiaret University	President
Mr. MEGHAZI Hadj	MAA	Tiaret University	Examiner
Mrs. BENATHMANE Lalia	MAA	Tiaret University	Supervisor

Academic year 2019/2020

Acknowledgment

First of all, I would like to thank the good Lord the Almighty for giving me the courage and the will to carry out this project.

I want to thank my dear parents who have supported and encouraged me throughout my life and during my studies.

It is with great pleasure that I keep these few lines as a sign of gratitude and deep gratitude to all those who, near or far, have contributed to the completion and crowning of this work.

I would like to thank Madame BENOTMANE Lalia for her support, her seriousness, her kindness and above all for having encouraged her to develop this work.

I sincerely thank Mr. OUARED for his advice and assistance throughout these years. It really deserves my thanks and gratitude.

I also want to express the honor bestowed on me by the members of the jury, by agreeing to judge my work.

I also want to thanks my friends for helping me in this work

Dedications

TO MY DEAR PARENTS

That no dedication can express what I owe them, for their kindness, affection and support ... Treasures of kindness, generosity and tenderness, in testimony of my deep love and my great gratitude "May God keep you".

TO MY SISTERS AND MY BROTHER

As a testament to my sincere appreciation for their efforts in the completion of my studies. I dedicate this modest work to them as a testimony of my great love and my infinite gratitude.

TO MY FRIENDS

For their help and moral support during the development of the work of end of study.

TO ALL MY FAMILY

To all those whose forgetting the name is hardly that of the heart ...

« There is no such thing as an" if "or" but "you have to succeed »

LENOUAR Miloud

Table of Contents

List of Figures

List of Tables

General Introduction

At this time, there is a tremendous increase in information available on the Internet level on many sites and the process of searching for information has become very expensive, prompting many users and companies, in particular, to resort to processes that help their customers find this information quickly. This means, recommendation systems that have become widely used in many places, and this method is divided into several sections, including recommendation systems that depend on user behavior such as collaborative filtering, and some of them depend on the type of elements that were searched such as content-based filtering.

In the first type of these systems, we face many problems, among them the cold start problem which is when we receive a new user or a new item, and the data dispersion problem, which is the presence of a small number of data for the user - the element array.

The problem here is how we can fill these blanks states with correct values, so there are many solutions available such as KNN, SVM, BN, MF, but we chose two algorithms: K-Nearest Neighborhood (KNN) and Matrix Factorization (MF) and we implemented on a dataset and we compare between it.

Our work in this research has been divided as follows:

In the first chapter, we discuss the collaborative filtering system and the most important problems in it, then in the second chapter we chose one of these problems which is the subject of our research in this thesis and we defined it and presented the proposed solutions to solve this problem, and in the last chapter with the help of the machine learning, we have implemented the two algorithms in order to solve the mentioned problem, these algorithms are among the most recent technologies currently used in the recommendation system.

CHAPTER1: Collaborative Filtering

CHAPTER1: Collaborative Filtering

1.1. Introduction :

In its recent sense, collaborative filtering is underlying recommendation systems. It brings together techniques which aim to make a selection on the elements to be presented to users (filtering) based on the behavior and expressed tastes of a large number of other users (collaboration) **(Wikipedia).**

Information gathering plays a crucial role in the process, it can be:

- **Explicit:** The user assigns notes to the product or indicates his appreciation (like).
    - Advantages: no ambiguities about the tastes and interests of the user.
    - Disadvantages: reporting bias, often exaggeration.
- **Implicit:** Behavior-based collection (purchases, clicks, page duration).
    - Advantages: objectivity.
    - Disadvantages: large volume, no indication of the appreciation

1.2. General Architecture of collaborative filtering[1]:

The general architecture of a collaborative filtering system revolves around two central entities: the user communities, and the list of evaluations of documents by the community that is used to issue recommendations, **Figure 1** shows each user. Belonged to a specific community, for a document D to be sent to user A in community B, D must accumulate a certain level of approval from members of B.

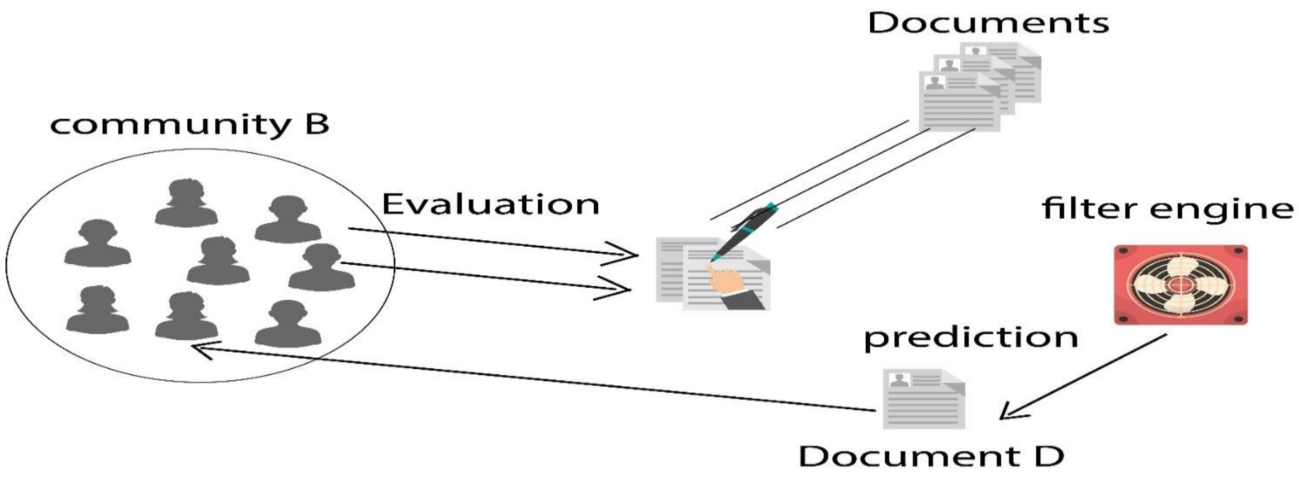


Figure 1: the general architecture of a collaborative filtering system

1.3. Operation process[1]:

There are three main processes in a collaborative filtering system which are [2]:

- **The formation of communities:** Performed by the system each time profiles are updated or a new user arrives (the model generation phase).
- **Production of recommendations:** Executed by the system when new information arrives (reformation of communities, recommendation phase etc.).
- **Recommendation review:** Performed by user, upon receipt of a recommendation.

We can see this process in the Figure 2 bellow:

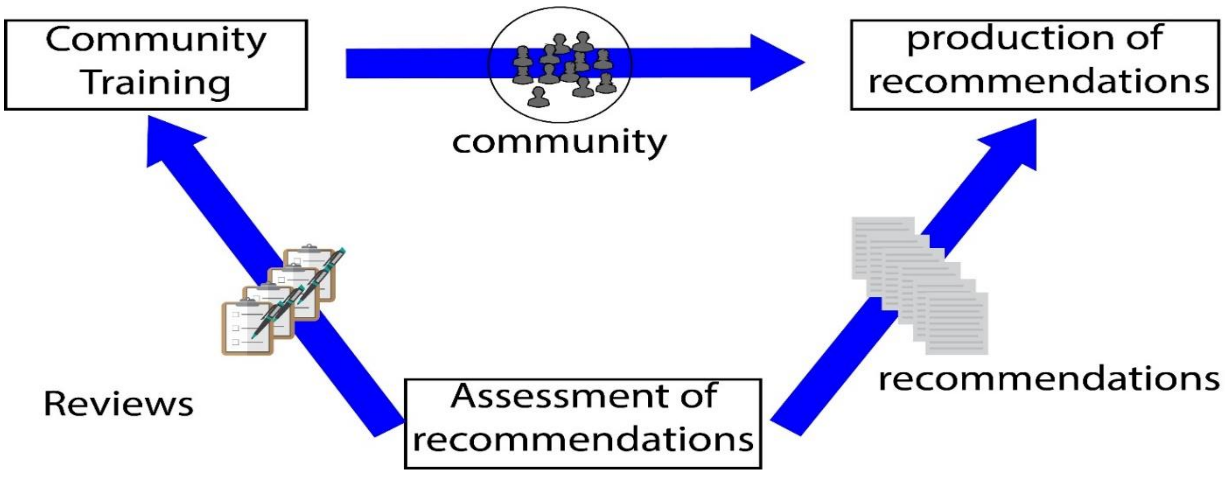


Figure 2: The three main processes of collaborative filtering

1.3.1. The formation of communities[1]

1.3.1.1. Definition

The notion of community in a collaborative filtering system is defined as the grouping of users according to the history of their evaluations, so that the system calculates recommendations [1].

The formation of communities is the core of a collaborative filtering system, helps in the production of recommendations, consists in grouping the users who have common properties, for this several approaches are used, which can be classified into two categories according to the type of the preferences of the user.

1.3.1.2. The issues of the formation of communities:

The community formation phase is the basis of the collaborative filtering process to influence the quality of recommendations sent to users, each time this process is triggered due to a new user who requests to join the system or the profiles of a user are changed, three problematic aspects are often to be addressed:

- **Perception of communities for users:** This situation can lead to dissensions such as:
  - ➢ A user does not agree with the recommendations sent over time by the system.
  - ➢ A user wants to choose a community himself.
  - ➢ A user wants to change the community chosen by the system.

- **Single-criteria formation of communities:**

  It is the problem of the single-criteria formation of communities by the history of evaluations in traditional collaborative filtering systems, which limits the enrichment and diversification of recommendations for users.

- **Positioning of users in communities:**

  The positioning of users in the communities depends fundamentally on the quality of the values given for each user on each criterion. The lack of value for one or more criteria leads to a difficulty in positioning users in communities, and in choosing the right community. The two major problems that can arise:

Cold Start Problem:

Cold start is the phenomenon that occurs early in the use of the system, in critical situations where the system lacks data to perform good quality personalized filtering [3], it is thus unable to recommend documents to users until they have enough information about their preferences and areas of interest.

➢ **Sparse Matrix:**

In collaborative filtering, only the objects to recommend are described by the ratings provided by the users. But in reality, it is impossible to force users to rate recommendations, as we can have a user with few rated resources generally, in which case we cannot get them if the user is in a better community or not.

Since this is the subject of our thesis. In the next section of the thesis, we will detail this problem and address the solutions that exist to solve this problem.

1.3.2. Producing recommendations

We can define this process as a Boolean function with two parameters, document, and user. a "d" document will be recommended to the user "u" who is integrated into the "G" community if this document is appreciated by the "G" community, users closest to "u" gave a favorable value judgment. This process is generally triggered for two reasons: upon arrival of a "new item" document or a new user is integrated into a "new user" community.

For the production of recommendations to an active user "u", the system predicts the interest of each document evaluated by the community members of "u", when it exceeds a certain threshold, the system recommends the document to the active user. Prediction computational techniques can be classified into three broad categories: “Memory-based algorithms”, “Model-based algorithms”.

1.3.2.1. Memory-based Approach to Collaborative Filtering

A straightforward algorithmic approach to collaborative filtering involves finding k nearest neighbors (i.e. the most similar users) of the active user and averaging their ratings of the item in question. Even more, we can calculate the weighted average of the ratings – weights being similarity, correlation or distance factors (later on in the text, the term “similarity” is used to denote any of the three measures) between a neighbor user and the active user like J. S. Breese, D. Heckerman, and C. Kadie do [4] . We can look at a user as being a feature vector. In this aspect, items that are being rated are features and ratings given by the user to these items are feature values. The following formula can be applied to predict user u’s rating of item i:

$$p_{u,i} = \bar{v}_u + \kappa \sum_{j \in U\text{sers}} w(u,j)\left(v_{j,i} - \bar{v}_j\right) \qquad (1)$$

Equation 1 was introduced by P. Resnick, N. Iaocvou, M. Suchak, P. Bergstrom, and J. Riedl [5]. where w(u1, u2) is the weight which is higher for more similar, less distant or more correlated users (feature vectors), $\bar{v}_u$ is the mean rating given by user u, and $\bar{v}_j$ is the mean rating given by user j $v_{j,i}$ is the rating of item i given by user j, and

κ is merely a normalization factor that depends on our choice of weighting.If no ratings of item i are available, the prediction is equal to the average rating given by user u. This is an evident improvement to the equation that simply calculates the weighted average.

Weight Computation: The weights can be defined in many different ways. Some of the possibilities are summarized in the following paragraphs.

1.3.2.1.1. Pearson Correlation:

The weights can be defined in terms of the Pearson correlation coefficient [5]. Pearson correlation is primarily used in statistics to evaluate the degree of linear relationship between two random variables. It ranges from −1 (a perfect negative relationship) to +1 (a perfect positive relationship), with 0 stating that there is no relationship whatsoever. The formula is as follows:

$$w(u_1,u_2)=\frac{\sum_{j\in\text{Items}}(v_{u_1,j}-\bar{v}_{u_1})(v_{u_2,j}-\bar{v}_{u_2})}{\sqrt{\sum_{j\in\text{Items}}(v_{u_1,j}-\bar{v}_{u_1})^2\sum_{j\in\text{Items}}(v_{u_2,j}-\bar{v}_{u_2})^2}} \qquad (2)$$

where $\bar{v}_{u_x}$ is the average rating of user X over all the items she rated, and $j \in$ Items is the set of items rated by both u_1 and u_2.

1.3.2.1.2. Cosine Similarity:

The similarity measure can be defined as the cosine of the angle between two feature vectors [6]. This technique is primarily used in information retrieval for calculating similarity between two documents, where documents are usually represented as vectors of word frequencies. In this context, the weights can be defined as:

$$w(u_1,u_2)=\sum_{i\in\text{Items}}\frac{v_{u_1,i}\,v_{u_2,i}}{\sqrt{\sum_{k\in I_1}v_{u_1,k}^2}\sqrt{\sum_{k\in I_2}v_{u_2,k}^2}} \qquad (3)$$

1.3.2.1.3. The Jaccard index:

The Jaccard index, in turn, is defined as the size of the intersection divided by the size of the union of two profiles, regardless of the rating associated to items, making this metric computationally efficient [6].

$$\text{Jaccard}(A, N) = \frac{|r_{u_1} \cap r_{u_2}|}{|r_{u_1} \cup r_{u_2}|} \qquad (4)$$

1.3.2.1.4. Memory-Based Advantages:

- Easy implementation
- New data can be added easily and incrementally
- Need not consider the content of items being recommended
- Scales well with correlated items

1.3.2.1.5. Memory-Based Limitations:

- Are dependent on human ratings
- Cold start problem for new user and new item
- Sparsity problem of rating matrix
- Limited scalability for large datasets

1.3.2.2. Model-based Approaches to Collaborative Filtering

In contrast to a memory-based method, a model-based method first builds a model out of the user-item matrix. The model enables faster and more accurate recommendations. In the following paragraphs we present two different approaches to model-based collaborative filtering. In the first one, we perceive collaborative filtering as a classification task. We employ a supervised learning algorithm (we'll explain it in detail in Chapter 3) to build a model. In the second one, we are concerned with reducing the dimensionality of the initial user-item matrix and thus build a model that is a lower-dimensional representation of the initial user-item database.

1.3.2.2.1. Collaborative Filtering as a Classification Task:

The collaborative filtering task can also be interpreted as a classification task, classes being different rating values [6]. Virtually any supervised learning algorithm can be applied to perform classification (i.e. prediction). For each user we train a separate

classifier. A training set consists of feature vectors representing items the user already rated, labels being ratings from the user. Clearly the problem occurs if our training algorithm cannot handle the missing values in the sparse feature vectors. It is suggested by [3] to represent each user by several instances (optimally, one instance for each possible rating value). On a 1–5 rating scale, user A would be represented with 5 instances, namely A-rates-1, A-rates-2... A-rates-5. The instance A-rates-3, for example, would hold ones ("1") for each item that user A rated 3 and zeros ("0") for all other items. This way, we fill in the missing values. We can now use such binary feature vectors for training. To predict a rating, we need to classify the item into one of the classes representing rating values. If we wanted to predict ratings on a continuous scale, we would have to use a regression approach instead of classification.

1.3.2.2.2. Dimensionality Reduction Techniques:

We are initially dealing with a huge user-item matrix. Since there can be millions of users and millions of items, the need to reduce the dimensionality of the matrix emerges. The reduction can be carried out by selecting only relevant users (instance selection) and/or by selecting only relevant items (feature selection). Other forms of dimensionality reduction can also be employed. It is shown by some researchers that feature selection, instance selection and other dimensionality reduction techniques not only counter the scalability problem but also result in more accurate recommendations [6, 7, 8]. Furthermore, the sparsity of the data is consequentially decreased.

When reducing the dimensionality, the first possibility that comes to mind is removing the users that did not rate enough items to participate in collaborative filtering. From the remaining users, we can randomly choose n users to limit our search for the neighborhood of the active user. This method is usually referred to as random sampling. Also, rarely rated items can be removed for better performance. Still, these relatively simple approaches are usually not sufficient for achieving high scalability and maintaining the recommendation accuracy

1.3.2.2.3. Latent Semantic Analysis (LSA):

A more sophisticated dimensionality reduction approach is called Latent Semantic Analysis (LSA) [8]. It is based on Singular Value Decomposition (SVD) of the user-item matrix. By using linear algebra, we can decompose a matrix into a triplet, namely $\boldsymbol{M} = \boldsymbol{U\Sigma V}^T$ the diagonal matrix $\boldsymbol{\Sigma}$ holds the singular values of $\boldsymbol{M}$ If we set all but K largest singular values to zero and thus obtain $\boldsymbol{\Sigma'}$, we can approximate $\boldsymbol{M}$ as $\boldsymbol{M'} = \boldsymbol{U\Sigma' V}^T$ By doing so, we transform our initial high-dimensional matrix into a K-dimensional (low dimensional) space. The neighborhood of the active user can now be determined by transforming the user vector into the low-dimensional space of the approximated matrix and finding k nearest points representing other users. Searching through a low-dimensional space clearly demands less time. Furthermore, dimensionality reduction reduces sparsity and captures transitive relationships among users. This results in higher accuracy.

1.3.2.2.4. Probabilistic Latent Semantic Analysis (pLSA):

On the basis of LSA, Probabilistic Latent Semantic Analysis (pLSA) was developed by T. Hofmann [11]. pLSA has its roots in information retrieval but can also be employed for collaborative filtering [12]. In a statistical model, an event like “person u ‘clicked on’ item i” is presented as an observation pair (u, i) (note that in such case we are dealing with implicit ratings). User u and item i “occur” paired with a certain probability: P(u, i). We are in fact interested in the conditional probability of item i occurring given user u: P (i|u). This conditional form is more suitable for collaborative filtering since we are interested in the active user’s interests.

The main idea of an aspect model (such as pLSA) is to introduce a latent variable z, with a state for every possible occurrence of (u, i). User and item are rendered independent, conditioned on z: $P(u \mid i)$ = $P(z)\ P(u \mid z)\ P(i \mid z)$. $P(i \mid u)$ Can be written in the following form:

$$P(i \mid u) = \sum_z P(i \mid z)P(z \mid u) \qquad (4)$$

In (4), the probabilities $P(i \mid z)$ and $P(z \mid u)$ can be determined by the Expectation Minimization (EM) algorithm using various mixture models. To support explicit ratings, we extend pLSA by incorporating ratings to our observation pairs and thus observing triplets of the form(u, i, r), where r represents a rating value.

Note that we limit the number of different states of z so that it is much smaller than the number of (u, i) pairs. Let us denote the number of users with Nu, the number of items with Ni, and the number of different states of z with N_z, where N_z << N_u, N_i. We can describe the probabilities $\mathrm{P}(i|u)$ with $S_1 = N_i \times N_u$ independent parameters. On the other hand, we can summarize the probabilities $\mathrm{P}(i|z)$ and $\mathrm{P}(z|u)$ with [11]:

$$S_2 = N_i \times N_z + N_u \times N_z.$$

The dimensionality reduction is evident from the fact that $S_2 < S_1$ (if N_z is small enough). Such latent class models tend to combine items into groups of similar items, and users into groups of similar users. In contrast to clustering techniques (see Sect. **1.2.4.2**), pLSA allows partial memberships in clusters (clusters being different states of z).

The relation of this method to LSA and SVD can be explained by representing the probabilities $P(i \mid u)$ in the form of a matrix $\boldsymbol{M_p}$ which can be decomposed into three matrices, namely $\boldsymbol{M_p = U_p \Sigma_p V_p^T}$ Described in the section (**1.2.2.3**).

1.3.2.2.5. Model-Based Advantage

- Better addresses the sparsity and scalability problem
- Improve prediction performance

1.3.2.2.6. Model-Based Limitations

- Expensive model building
- Trade-off between the prediction performance and scalability
- Loss of information in dimensionality reduction technique (SVD)

1.3.2.3. Item-based Collaborative Filtering

All collaborative filtering approaches that we have discussed so far, are user-centric in the way that they concentrate on determining the user's neighborhood. Some researchers also considered item-based collaborative filtering such as B. Sarwar, G. Karypis, J. Konstan, and J. Reidl [12]. The main idea is to compute item-item similarities (according to the users' ratings) offline and make use of them in the online phase.

To predict user u's rating of item i, the online algorithm computes a weighted sum of the user u's ratings over k items that are most similar to item i. The main question in this approach is how to evaluate item-item similarities to compute a weighted sum of the ratings. Item-item similarities can be computed by using the techniques for computing user-user similarities, described in Sect. **1.3.1.2** the winning technique, according to [12], is the so called adjusted cosine similarity measure. This is a variant of cosine similarity which incorporates the fact that different users may have different rating scales. The similarity measures are then used as weights for calculating a weighted sum of k nearest items.

1.3.2.4. Some Other Approaches

Let us briefly summarize some other techniques. Interested reader should consider the appropriate additional reading.

1.3.2.4.1. Clustering Techniques:

Bayesian and non-Bayesian clustering techniques can be used to build clusters (or neighborhoods) of similar users [4, 6, 12]. The active user is a member of a certain cluster. To predict his/her rating of item i, we compute the average rating for item i within the cluster that the user belongs to. Some such methods allow partial membership of the user in more than one cluster. In such case, the predicted rating is summed over several clusters, weighted by the user's participation degree. Clustering techniques can also be used as instance selection techniques (instances being users) that are used to reduce the candidate set for the k-Nearest Neighbors algorithm.

1.3.2.4.2. Bayesian Networks:

Bayesian networks with a decision tree at each node have also been applied to collaborative filtering [13]. Nodes correspond to items, and states of each node correspond to possible rating values. Conditional probabilities at each node are represented as decision trees in which nodes again are items, edges represent preferences, and leaves represent the possible states (i.e. rating values). Bayesian networks are built offline over several hours or even days. This approach is not suitable in systems that need to update rapidly and frequently.

1.4. Collaborative Filtering Data Characteristics

As already mentioned, the data in the user-item interaction database can be collected either explicitly (explicit ratings) or implicitly (implicit preferences). In the first case, the user's participation is required. The user is asked to explicitly submit his/her rating for the given item. In contrast to this, implicit preferences are inferred from the user's actions in the context of an item (that is why the term "user-item interaction" is used instead of the word "rating" when referring to users' preferences in the following sections). Data can be collected implicitly either on the client side

or on the server side. In the first case, the user is bound to use modified client-side software that logs his/her actions. Since we do not want to enforce modified client-side software, this possibility is usually omitted. In the second case, the logging is done by a server. In the context of the Web, implicit preferences can be determined from access logs that are automatically maintained by Web servers.
Collected data is first preprocessed and arranged into a user-item matrix. Rows represent users and columns represent items. Each matrix element is in general a set of actions that a specific user took in the context of a specific item. In most cases a matrix element is a single number representing either an explicit rating or a rating that was inferred from the user's actions.

Since a user usually does not access every item in the repository, the vector (i.e. the matrix row) representing the user is missing some/many values. To emphasize this, we use the terms "sparse vector" and "sparse matrix".

The fact that we are dealing with a sparse matrix can result in the most concerning problem of collaborative filtering – the so called sparsity problem. In order to be able to compare two sparse vectors, similarity measures require some values to overlap. What is more, the lower the amount of overlapping values, the lower the reliability of these measures. If we are dealing with a high level of sparsity, we are unable to form reliable neighborhoods. Furthermore, in highly sparse data there might be many unrated (unseen) items and many inactive users. Those items/users, unfortunate l y, cannot participate in the collaborative filtering process [14].

Sparsity is not the only reason for the inaccuracy of recommendations provided by collaborative filtering. If we are dealing with implicit preferences, the ratings are usually inferred from the user-item interactions, as already mentioned earlier in the text.

Mapping implicit preferences into explicit ratings is a non-trivial task and can result in false mappings. The latter is even truer for server-side collected data in the context of the Web since Web logs contain very limited information. To determine how much time a user was reading a document, we need to compute the difference in time- stamps of two consecutive requests from that user. This, however, does not tell us whether the user was actually reading the document or he/she, for example, went out to lunch, leaving the browser opened. What is more, the user may be accessing cached information (either from a local cache or from an intermediate proxy server cache) and it is not possible to detect these events on the server side [14].

1.5. Advantages and disadvantages of collaborative filtering

Collaborative filtering methods have several advantages, the most important of which are:

- Surprise effect (finding something other than what you were looking for).
- Domain knowledge notrequired.
- Possibility of indexing all kinds of items.
- Elimination of the problem of over-specialization.
- The quality of the suggestions improves over time.

However, the use of collaborative filtering techniques can lead to several problems:

- Cold start (a new user who has not noted any item cannot receive a recommendation since the system does not know his tastes).

Sparse Matrix: when the user-item matrix contains a few items which have a rating value (the subject of our thesis we will discuss it later).

- Parsimony (sparsity, the number of candidate items for recommendation is often huge and users only rate a small subset of the items available).
- The gray sheep problem will not have many neighboring users, so it will be difficult to make relevant recommendations for such users.

1.6. Examples of collaborative filtering systems:

A few years later, with the rise of the Internet and Web applications, there was a craze for recommendation systems and especially collaborative ones that developed in different application areas. We can cite some:

- **Amazon**

Amazon.com uses the “item-to-item collaborative filtering” algorithm [15]. This system starts with the calculation of the degree of similarity between offline articles, thus constructing a table of article similarities. This step is extremely demanding in terms of computation time. Then, if the user is interested in a specific product, the system recommends products similar to this one based on the item similarity matrix.

- **COCOFil**

The COCoFil platform [16] (Community-Oriented Collaborative Filtering) comprises three modules: Collaborative filtering, Configuration, and Contact management. The "contact management" module ensures identification in the COCoFil platform, that is to say, it allows users on the one hand to enter their personal information, and on the other hand for the system to identify the person. the user when accessing it. Thanks to this module, the user can also organize their address book and exchange recommendations with other users as part of active collaborative filtering.

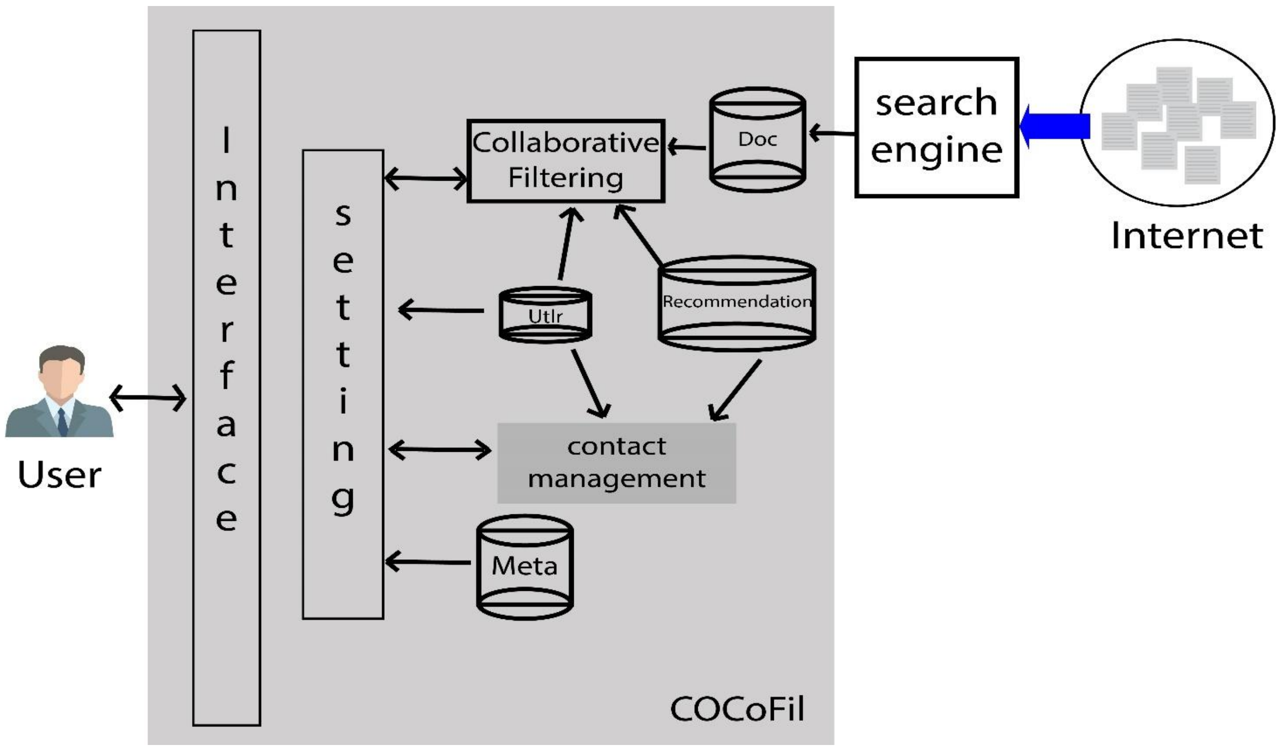


Figure 3: The architecture of the COCoFil platform

- **GroupLens**

GroupLens [17], is an experimental system from the University of Minnesota, it is one of the most famous and strong in this field. It is similar in spirit to Tapestry: readers are asked to rate the articles they read. The system then finds correlations between different users and identifies groups of users with similar interests. Then, it uses these estimates to predict the interest users will have in each article.

- **Tapestry**

The concept of collaborative filtering was started with the Tapestry project at Xerox Parc. Managing e-mails is his primary motivation. Tapestry is based on a "commented recommendation" based on quality ratings or document appreciation made by users. In this way, the documents are filtered according to these annotations [16].

- **Netflix Prize**

The Netflix Prize was an open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings without any other information about the users or films, i.e. without the users or the films being identified except by numbers assigned for the contest.
The competition was held by Netflix, an online DVD-rental and video streaming service,

and was open to anyone who is neither connected with Netflix (current and former employees, agents, close relatives of Netflix employees, etc.).

1.7. Conclusion

In this Chapter we see the collaborative filtering technique and there types and discussing each type of it and see some examples of this technique.

CHAPTER 2: Machine Learning and Sparse Matrix Problem

CHAPTER2: Machine Learning and Sparse Matrix Problem

2.1. Introduction

As we mentioned earlier, the problem with a sparse matrix is that there are few cells that contain values, which makes the process of calculating the similarity between users or items very difficult.

In this section, we will talk about the solutions to this problem and in the next section, we will program these solutions with the help of machine learning.

2.2. Sparse Matrix

When providing recommendations to the user, we can find out how satisfied the user is with the recommendations by evaluating them, but he is not obligated to do so. And this last thing is the main reason for the emergence of this problem

With a huge number of users and things, the process of calculating the similarity value between users to make recommendations becomes very difficult and almost impossible.

And in this case, the user-item matrix will appear as shown in the **Figure 4** [10] below:

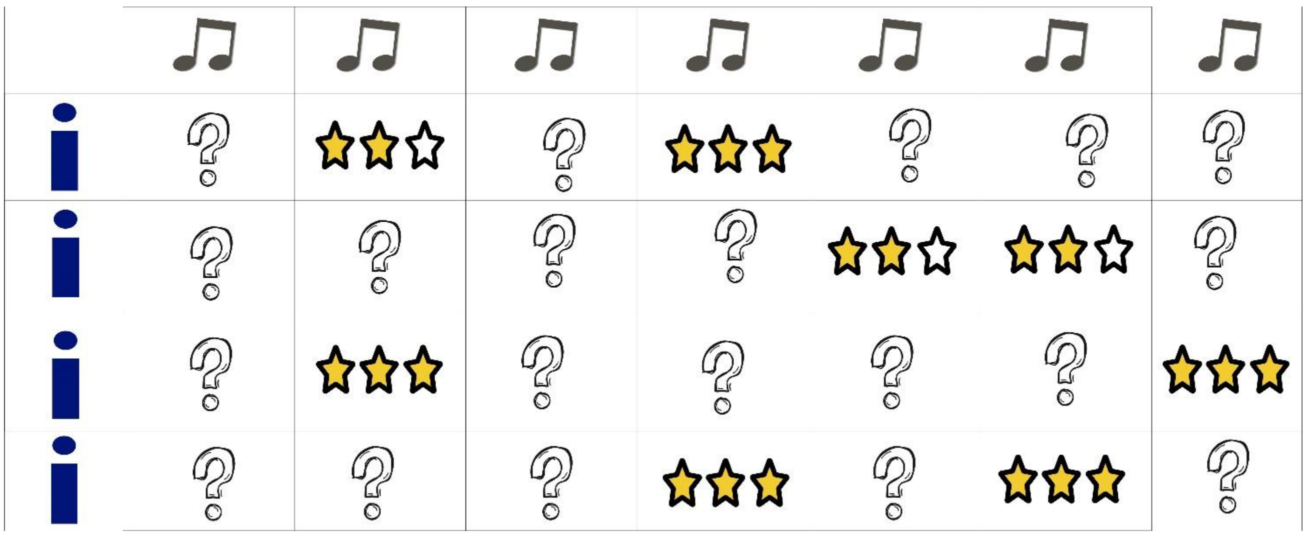


Figure 4: Sparse Matrix preview

2.3. Proposed Solutions for solving this problem

After a long research in this field, we found many algorithms that work to solve this problem and to program these solutions such as SVM, KNN, MF, BN, we choose KNN and MF and with the help of Machine Learning we implement it on the dataset and compare between it.

For this we will go into the definition of machine learning and show its characteristics

- **Machine learning Definition**

The ability to learn from past experiences and to adapt is an essential characteristic of higher life forms. It is essential for humans in the early stages of life to learn such basic things as recognizing a voice, a familiar face, learning to understand what is said, to walk and to speak [11].

Machine learning is an attempt to understand and reproduce this learning ability in artificial systems. It is a question, very schematically, of designing algorithms capable, from a large number of examples (the data corresponding to “past experience”), of assimilating its nature in order to be able to apply what they thus learned about future cases [12].

- **Concepts and Sources of Machine Learning**

Human learning is made up of several processes that are difficult to describe precisely. The learning abilities in humans have given them an evolutionary advantage which is decisive for their development [13].

By "ability to learn" is meant a set of skills such as:

- Obtaining the ability to speak by observing others.
- Obtaining the ability to read, write, and perform arithmetic and logic operations with the help of a tutor.
- Obtaining motor and sports skills by exercising.

Machine learning of a machine always concerns a set of concrete tasks T. To determine the performance of the machine, a performance measure is used P. The machine may have in advance a set of experience E or she will enrich this set later.

So, machine learning for the machine is that with the set of tasks T that the machine has to perform, it uses the set of experiences E such that its performance on T is improved.

- **Application areas of machine learning:**

Machine learning is applicable to a large number of human activities and is particularly suited to the problem of automated decision making. This will be, for example:

- To establish a medical diagnosis from the clinical description of a patient.
- To respond to a client's request for a bank loan based on his personal situation.
- To trigger an alert process based on signals received by sensors.
- Pattern recognition.
- Recognition of speech and written text.
- Control a process and diagnose failures.

Types of learning

- **Supervised learning**

If the classes are predetermined and the examples are known, the system learns to classify according to a classification model; we then speak of supervised learning (or discriminant analysis).

An expert (or oracle) must first correctly label examples. The learner can then find or approximate the function which makes it possible to assign the right “label” to these examples. Sometimes it is preferable to associate a piece of data not with a single class, but with a probability of belonging to each of the predetermined classes (this is called probabilistic supervised learning). Typical examples are linear discriminant analysis or SVMs. Another example: based on common points detected with the symptoms of other known patients (the “examples”), the system can categorize new patients on the basis of their medical analyzes as estimated risk (probability) of developing a particular disease.

- **Unsupervised learning**

When the system or the operator has only examples, but not labels and the number of classes and their nature has not been predetermined, we speak of unsupervised learning (or clustering). No expert is available or required. The algorithm must discover by itself the more or less hidden structure of the data.

The system must here in the description space (the sum of the data) target the data according to their available attributes, to classify them in homogeneous groups of examples. The similarity is generally calculated according to the function of the distance between pairs of examples. It is then up to the operator to associate or deduce meaning for each group. Various math tools and software can help him. We also speak of regression data analysis. If the approach is probabilistic (that is to say that each example instead of being classified in a single class is associated with the probabilities of belonging to each of the classes), we then speak of "soft clustering" (as opposed to “Hard clustering”) [13].

Example: An epidemiologist could, for example, in a fairly large group of victims of liver cancer try to bring out explanatory hypotheses, the computer could differentiate

different groups, which we could then associate for example with their geographical origin, genetic, alcoholism, or exposure to heavy metal or toxin such as aflatoxin.

- **Semi-supervised learning**

Carried out in a probabilistic or non-probabilistic manner, it aims to show the underlying distribution of “examples” in their description space. It is implemented when data (or “labels”) is missing ... The model must use unlabeled examples that can nevertheless provide information.

Example: In medicine, it can be an aid in the diagnosis or in the choice of the least expensive means of diagnostic tests.

- **Partially supervised learning (probabilistic or not)**

When data labeling is partial. This is the case when a model states that data does not belong to a class A, but perhaps to a class B or C (A, B, and C is 3 diseases for example mentioned in the context of a differential diagnosis).

- **Reinforcement learning**

The algorithm learns behavior from an observation. The action of the algorithm on the environment produces a return value that guides the algorithm.

- **The Algorithms used**

- Support vector machines
- Boosting
- Matrix Factorization
- Neural networks for supervised or unsupervised learning
- The k nearest neighbors method for supervised learning
- Decision trees
- Statistical methods such as the Gaussian mixture model
- Logistic regression
- Linear discriminant analysis

These methods are often combined to obtain various learning variants. The use of this or that algorithm strongly depends on the task to be solved (classification, estimation of values, etc.).

- **Relevance and efficiency factors**

The quality of learning and analysis depends on the upstream need and a priori competence of the operator to prepare the analysis. It also depends on the complexity of the model (specific or generalist) and its adaptation to the subject to be treated. Finally, the quality of the work will also depend on the mode (of visual highlighting) of the results for the end-user (a relevant result could be hidden in an overly complex diagram, or poorly highlighted by an inappropriate graphic representation). Before that, the quality of the work will depend on initial constraining factors, related to the database [14]:

1. **Number of examples:** the fewer there are, the more difficult the analysis, but the more there are, the greater the need for computer memory and the longer the analysis.
2. **Number and quality of attributes:** describing these examples (The distance between two numerical "examples" (price, size, weight, light intensity, noise intensity, etc.) is easy to establish, that between two categorical attributes (color, utility, is more delicate).
3. **Percentage** of completed and missing data.

4. **“Noise”:** The number and "location" of doubtful values (errors) or naturally not in conformity with the general distribution model of the "examples" on their distribution space

- Some classification methods

- **K nearest neighbors**

Better known in English as K-nearest neighbor (K-NN). This method differs from traditional learning methods because no model is inferred from the examples. The data remains as-is: it is simply stored in memory [14].
To predict the class of a new case, the algorithm looks for the K closest neighbors of this new case and predicts (if it is necessary to choose) the most frequent response of these K closest neighbors [14].

- **Decision trees**

Decision trees are the most popular of the learning methods. The known algorithms are ID3 (Quinlan 1986) and C4.5 (Quinlan 1993). Like any supervised learning method, decision trees use examples [14].

If we have to classify examples into categories, we must construct a decision tree by category. To determine to which category a new example belongs, we use the decision tree of each category to which we submit the new example to be classified.

Each tree responds Yes or No (it makes a decision). Concretely, each node of a decision tree contains a test (an IF ... THEN) and the leaves have the values Yes or No. Each test looks at the value of an attribute of each example. Indeed, it is assumed that an example is a set of attributes/values.

But this method has a drawback when the examples are limited: in fact, only 2/3 of the initial examples are used to build the tree, since 1/3 is reserved to validate it later. Many other methods have been proposed.

- **Naive or Bayes decisions**

Named after Bayes' theorem, these methods are called "Naïve" or "Simple" because they assume the independence of the variables. The idea is to use probability conditions observed in the data. We calculate the probability of each class among the examples.

A variant of the Naive Bayes is the Bayesian networks: in this model, we no longer assume that the variables are all independent, and we allow some to be linked.

This considerably increases the calculations and the results do not increase significantly.

- **Support Vector Machines (or SVM)**

This technique initiated by Vapnik attempts to linearly separate the positive examples from the negative examples in the set of examples. Each example must be represented by a vector of dimension n.

The method then searches for the hyperplane that separates the positive examples from the negative examples, ensuring that the margin between the closest positive and negative is maximum.

Intuitively, this guarantees a good level of generalization because new examples may not be too similar to those used to find the hyperplane but still be located frankly on one side or the other of the border.

As the state-of-the-art collaborative filtering recommender systems are based on two main approaches: K-Nearest Neighborhood (KNN) approach and latent factor models such as Matrix Factorization [15] we will choose these two algorithms to implement it.

2.3.1. K- Nearest Neighbors (KNN):

What is KNN? :

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well:

- **Lazy learning algorithm:** KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.

- **Non-parametric learning algorithm:** KNN is also a non-parametric learning algorithm because it doesn’t assume anything about the underlying data.

Moreover KNN is:

- A powerful classification algorithm used in pattern recognition [15].
- K nearest neighbors stores all available cases and classifies new cases based on a similarity measure (distance function).
- One of the top data mining algorithms used today.
- A non-parametric lazy learning algorithm (An Instance-based Learning method).

Classification Approach:

The Classification approach in the K-Nearest Neighbors (KNN) work as below:

- An object (a new instance) is classified by a majority vote for its neighbor classes.
- The object is assigned to the most common class amongst its K nearest neighbors. (measured by a distant function) [14]

Distance measure for KNN Algorithms:

- **Euclidean Distance**

Usually this algorithm uses Euclidean distance which is calculated in the following way [10]:

$$\text{Euclidean} \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

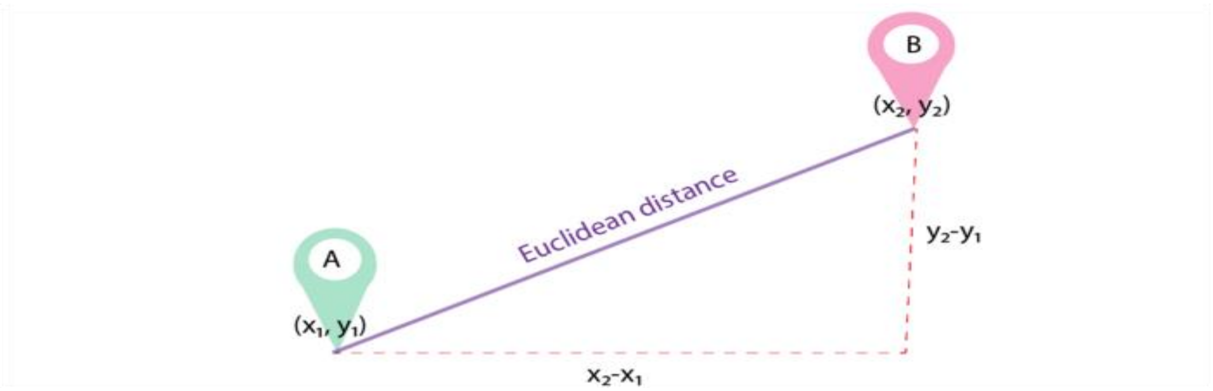


Figure 5: Euclidean distance

But there are other values that can be used to calculate the distance between users and they are [10]:

Manhattan Distance:

$$\text{Manhattan} \sum_{i=1}^{n} |x_i - y_i|$$

We use Manhattan distance, also known as **city block distance Figure 6**, or **taxicab geometry** if we need to calculate the distance between two data points in a grid-like path.[10]

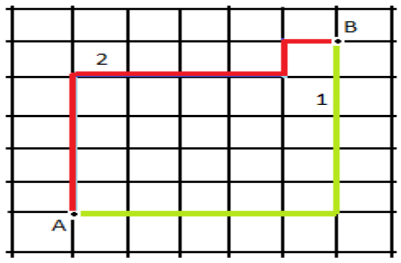


Figure 6: Manhattan distance

Minkowski Distance:

$$\text{Minkowski} \left(\sum_{i=1}^{n} (|x_i - y_i|)^q\right)^{1/q}$$

Minkowski distance is a generalized distance metric. We can manipulate the above formula by substituting ‘p’ to calculate the distance between two data points in different ways. Thus, Minkowski Distance is also known as L_p norm distance [10].

Some common values of ‘p’ are:

- p = 1, Manhattan Distance
- p = 2, Euclidean Distance
- p = infinity Chebyshev Distance

K-nearest neighbors (KNN) algorithm uses ‘feature similarity’ to predict the values of new data points which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps:

Step1: For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

Step2: Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step3: For each point in the test data do the following:

- Calculate the distance between test data and each row of training data with the help of any of the method shown above: Euclidean, Manhattan or Minkowski distance. The most commonly used method to calculate distance is Euclidean

$$\mathrm{sim}(u,u') = \sqrt{\sum_i \left(r_{u,i} - r_{u',i}\right)^2}$$

Where $r_{u,i}$ and $r_{u',i}$ are ratings of user u and user u' to movie i. And we choose top k most similar neighbors of each user under the above defined similarity function.

These values are arbitrary but important because very different results result from their choices. Note also that, if the learning time is nonexistent since the data is stored as-is, the classification of a new case is on the other hand costly since it is necessary to compare this case with all the examples already classified.

We use the weighted means to predict the ratings. The prediction formula is as follows:

$$P_{u,m} = \frac{\sum_{j \in N_u^K(m)} \mathrm{sim}(m,j) R_{j,u}}{\sum_{j \in N_u^K(m)} |\mathrm{sim}(m,j)|}$$

where N_i^K (m) = { j : j belongs to the K most similar user to user m and user u has rated j }, and $R_{j,u}$ are the existent ratings (of user u on movie j) and $P_{u,m}$ is the prediction.

Step4: End.

- **Pros and Cons of KNN**

Pros:

- It is very simple algorithm to understand and interpret.
- It is very useful for nonlinear data because there is no assumption about data in this algorithm.
- It is a versatile algorithm as we can use it for classification as well as regression.
- It has relatively high accuracy but there are much better supervised learning models than KNN.

Cons:

- It is computationally a bit expensive algorithm because it stores all the training data.
- High memory storage required as compared to other supervised learning algorithms.
- Prediction is slow in case of big N.
- It is very sensitive to the scale of data as well as irrelevant features.

The neighborhood-based methods can be divided into two different types, user-based and item-based collaborative filtering. This division is essentially just different ways of computing the neighborhood. In the user-based collaborative filtering methods only the users is used for computing the neighborhood and in item-based collaborative filtering only the items are used. For example, as the Figure below show in the user-based methods if user X and user Z have items in common that they like they are assumed to have similar taste in items that they do not have in common. On the other hand, in the item-based methods if user X like book A and book B and C are very similar to book A then user X is assumed to also like book B and C. When they are used differs on the context, but as an example, many e-commerce web-sites use item-based methods, while streaming sites for films and tv-series use both.

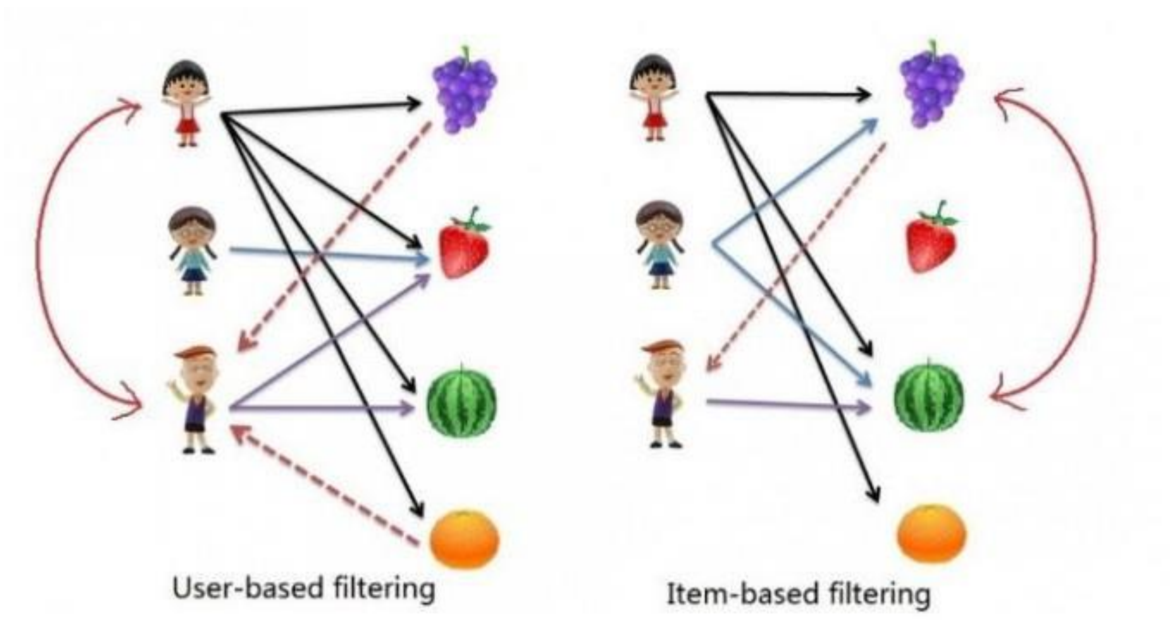


Figure 7: User Based and Item Based Filtering

2.3.2. Matrix Factorization (MF):

- **Overview**

Matrix factorization models (MF) became popular because of their scalability and their predictive performance [16] and was widely used in the Netflix competition solutions [17]. A probabilistic foundation for these models was then given by [18].

The general idea of matrix factorization in recommender systems is to create a low- rank matrix approximation of the rating matrix. The idea of low-rank matrix approximation was first proposed by Eckart and Young [19] and has been used in information retrieval since.

- **Working of Matrix Factorization(MF)**

Given a user u and a movie i, we predict the rating that the user will give to the movie as follows:

$$r_{u,i} = \mu + b_u + b_i + \gamma_u \cdot \gamma_i$$

Where μ is the global bias, and b_u (b_i) is the user (item) bias. γ_u And γ_i are latent factors for user u and movie i respectively, which will be learned during the training process. γ_u and γ_i are K dimensional vectors [20] .

The error function L is defined as follows [20]:

$$L = \sum_{u,i} \left(r_{u,i} - (\mu + b_u + b_i + \gamma_u \cdot \gamma_i) \right)^2 + \lambda_{ub} \sum_{u} \|b_u\|^2 + \lambda_{ib} \sum_{i} \|b_i\|^2 + \lambda_{u\gamma} \sum_{u} \|\gamma_u\|^2 + \lambda_{i\gamma} \sum_{i} \|\gamma_i\|^2$$

Whereλ_{ub} ,λ_{ib} ,$\lambda_{u\gamma}$,$\lambda_{i\gamma}$ are used to control the trade-off between accuracy and Complexity during training, and $\Sigma_u b_u^2, \Sigma_i b_i^2, \Sigma_u Y_u^2, \Sigma_i \gamma_i^2$ penalize model Complexity and reduces over-fitting.

We have the following expressions for the gradient of the error function:

$$\frac{\partial L}{\partial b_u} = 2\left(r_{u,i} - (\mu + b_u + b_i + \gamma_u \cdot \gamma_i) \right)(-1) + 2\lambda_{ub} b_u$$

$$\frac{\partial L}{\partial b_u} = 2\left(\mathrm{error}_{u,i}\right)(-1) + 2\lambda_{ub} b_u$$

$$\frac{\partial L}{\partial b_u} = -\mathrm{error}_{u,i} + \lambda_{ub} b_u$$

$$\frac{\partial L}{\partial b_i} = -\mathrm{error}_{u,i} + \lambda_{ib} b_i$$

$$\frac{\partial L}{\partial \gamma_u} = -\mathrm{error}_{u,i}\, \gamma_i + \lambda_{u\gamma} \gamma_u$$

$$\frac{\partial L}{\partial \gamma_i} = -\mathrm{error}_{u,i}\, \gamma_u + \lambda_{i\gamma} \gamma_i$$

There are two approaches to solving the above optimization problem to find all of our features: stochastic gradient descent (SGD) and alternating least squares (ALS) [20].

We use both SGD and ALS to update these parameters end up being:

$$b_u \leftarrow b_u + \eta\left(\text{error}_{u,i} - \lambda_{ub} b_u\right)$$
$$b_i \leftarrow b_i + \eta\left(\text{error}_{u,i} - \lambda_{ui} b_i\right)$$
$$\gamma_u \leftarrow \gamma_u + \eta\left(\text{error}_{u,i}\gamma_i - \lambda_{ub}\gamma_u\right)$$
$$\gamma_i \leftarrow \gamma_i + \eta\left(\text{error}_{u,i}\gamma_u - \lambda_{ub}\gamma_i\right)$$

Matrix factorization *assumes* that:

- Each user can be described by k attributes or features. For example, feature 1 might be a number that says how much each user likes sci-fi movies.
- Each item (movie) can be described by an analogous set of k attributes or features. Feature 1 for the movie might be a number that says how close the movie is to pure sci-f

If we multiply each feature of the user by the corresponding feature of the movie and add everything together, this will be a good approximation for the rating the user would give that movie.

2.4. Conclusion

In this section, we learned about each of the KNN and MF algorithms and how they work and how we can predict the messing value by them.

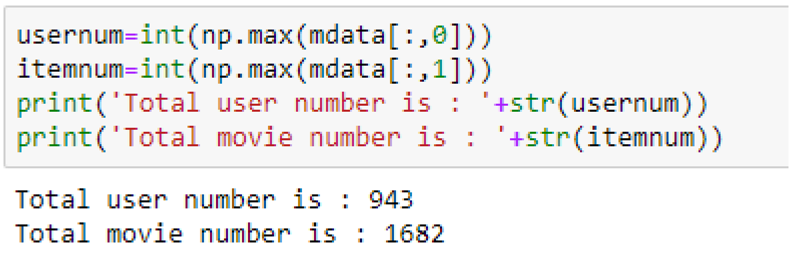
CHAPTER 3: Implementation of Solutions

3.1. Introduction

The goal of this assignment is to predict the rating given a user and a movie, using two different methods Shown above: K-Nearest Neighbors and Matrix Factorization method on the Movie Lens.

3.2. Data Description

The initial data set i.e. u.data, obtained from the Movie Lens, a movie recommendation service, has 100,000 ratings (1-5) from 943 users on 1682 movies as shown below:

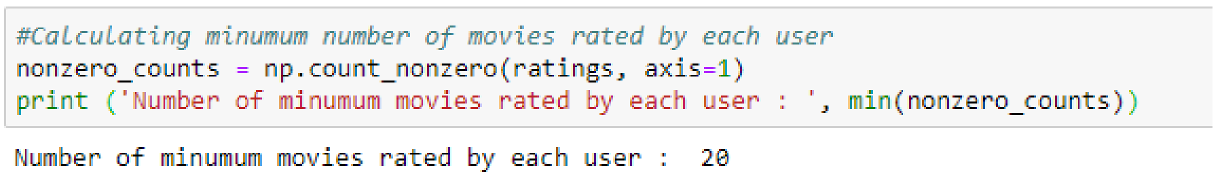


```
usernum=int(np.max(mdata[:,0]))
itemnum=int(np.max(mdata[:,1]))
print('Total user number is : '+str(usernum))
print('Total movie number is : '+str(itemnum))
```

```
Total user number is : 943
Total movie number is : 1682
```

Figure 8: the users and movies number

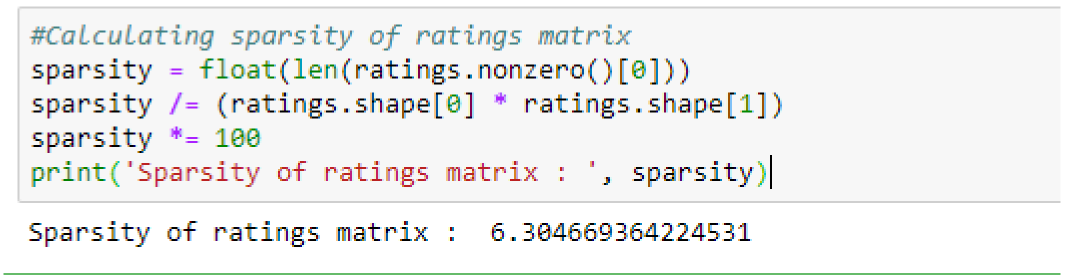
Feature Names were added to the Dataset. Dataset was loaded using python pandas and was transformed into a rating matrix represented as a NumPy array of size users x movies (943 x 1682). We found that users have rated at least 20 movies Figure 9 which result in a reasonable sparsity of 6.3% Figure 10. This means that 1,00,000 ratings account for only 6.3% of the total possible ratings.



```
#Calculating minumum number of movies rated by each user
nonzero_counts = np.count_nonzero(ratings, axis=1)
print ('Number of minumum movies rated by each user : ', min(nonzero_counts))
```

```
Number of minumum movies rated by each user :  20
```

Figure 9: movies rated by user



```
#Calculating sparsity of ratings matrix
sparsity = float(len(ratings.nonzero()[0]))
sparsity /= (ratings.shape[0] * ratings.shape[1])
sparsity *= 100
print('Sparsity of ratings matrix : ', sparsity)
```

```
Sparsity of ratings matrix :  6.304669364224531
```

Figure 10: Sparsity of our dataset

3.3. Metric to evaluate the Algorithms

Each sampled sparse dataset will be split in two sets, a training set and a testing set. The models are estimated on the training set and evaluated on the testing set For the testing set, predictions will be made with the models. Based on these predictions, we will measure the accuracy of the model. Several accuracy metrics have been used for measuring performance of recommender systems. Which are: RMSE, MAE, and Precision, Recall, F1-measure, Coverage, ROC curve, and AUC.

One of the most often used metrics is Root Mean Square Error (**RMSE**). It measures how close predicted ratings for recommended items, computed using a training subset of a dataset, and are to the actual ratings of items in the remaining testing subset of the same dataset, The Netflix prize used RMSE. It is defined as below for each user A:

$$RMSE = \sqrt{\frac{1}{|\hat{R}|} \sum_{(u,i) \in \tilde{R}} (r_{ui} - \hat{r}_{ui})^2}$$

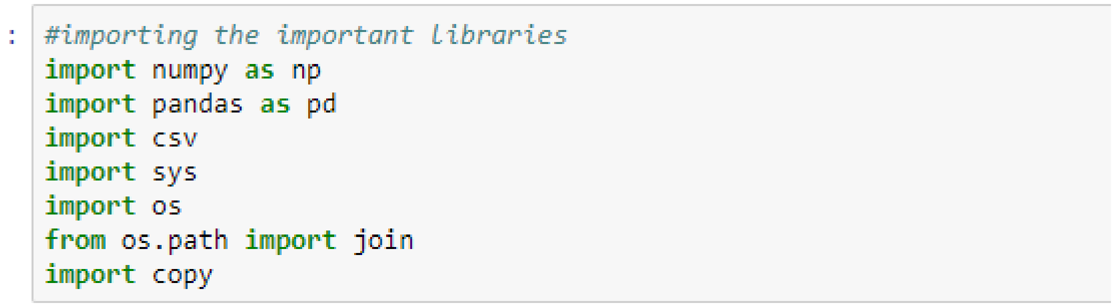
Where $|\hat{R}|$ is the number of ratings in the test set and (u, i) is the user-item pairs in this test set [21].

A related metric is Mean Absolute Error (**MAE**). It is similar to RMSE, except that it does not square differences between predicted and actual ratings, thus penalizing less harshly large prediction error, e.g. predicting a rating of 5 while the actual one is 1. MAE of user A is defined as follows:

$$MAE(A) = \frac{\sum_{i=1}^{n} |pred_{A,i} - r_{A,i}|}{n}$$

3.4. Implementation of KNN Algorithm

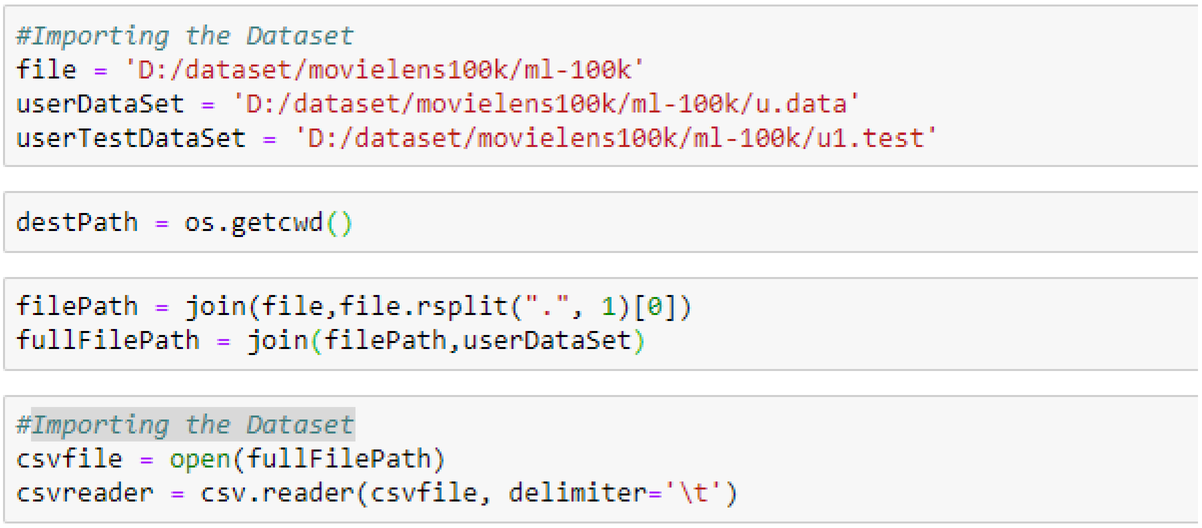
- First, we will be importing the important libraries that we work with



```
#importing the important libraries
import numpy as np
import pandas as pd
import csv
import sys
import os
from os.path import join
import copy
```

Figure 11: important libraries in python

- We import our dataset



```
#Importing the Dataset
file = 'D:/dataset/movielens100k/ml-100k'
userDataSet = 'D:/dataset/movielens100k/ml-100k/u.data'
userTestDataSet = 'D:/dataset/movielens100k/ml-100k/u1.test'
```

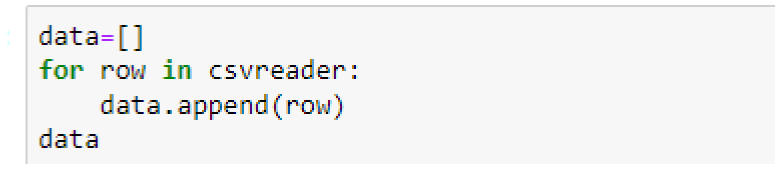
```
destPath = os.getcwd()
```

```
filePath = join(file,file.rsplit(".", 1)[0])
fullFilePath = join(filePath,userDataSet)
```

```
#Importing the Dataset
csvfile = open(fullFilePath)
csvreader = csv.reader(csvfile, delimiter='\t')
```

Figure 12: Fetching and importing the dataset

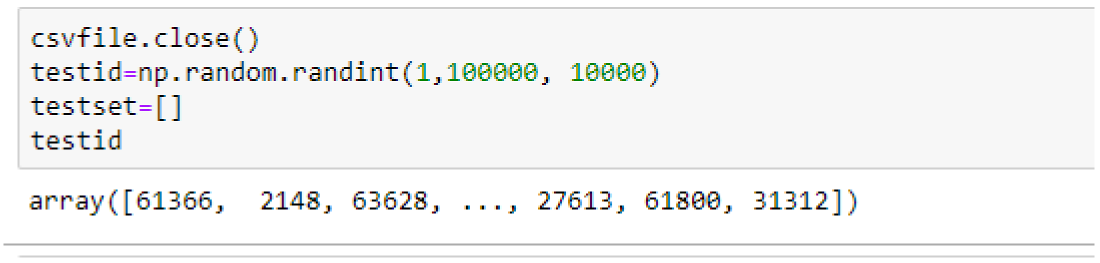
- We put a data in a simple array by using the append function



```
data=[]
for row in csvreader:
    data.append(row)
data
```

Figure 13: creating simple array with dataset data

- We create an array with a 10000 random number between 1 and 1000000

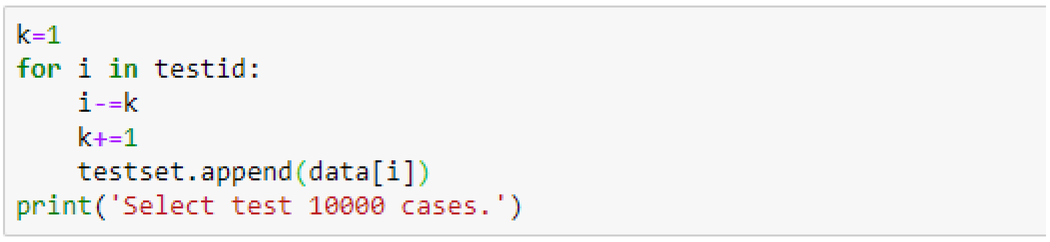


```
csvfile.close()
testid=np.random.randint(1,100000, 10000)
testset=[]
testid
```

```
array([61366,  2148, 63628, ..., 27613, 61800, 31312])
```

Figure 14: random number between 1 and 100000

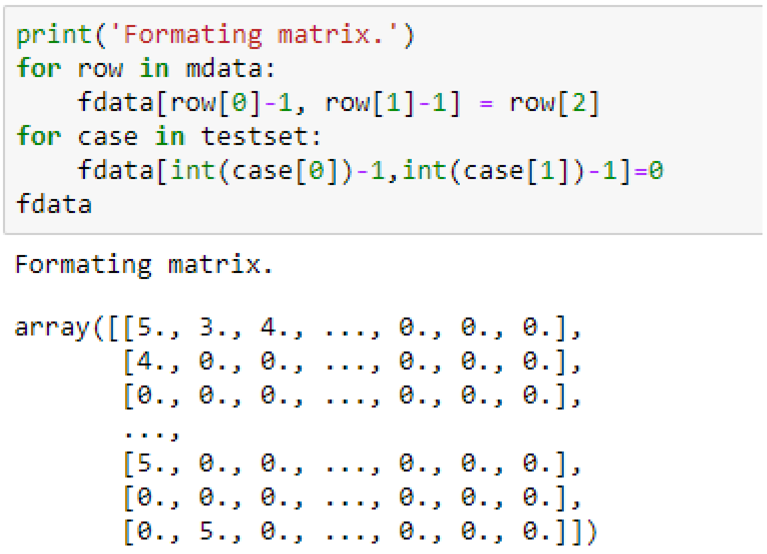
- Fill the testset array with the data corresponding to the random id in testid array



```
k=1
for i in testid:
    i-=k
    k+=1
    testset.append(data[i])
print('Select test 10000 cases.')
```

Figure 15: fill the testset array

- Formatting the matrix from the data



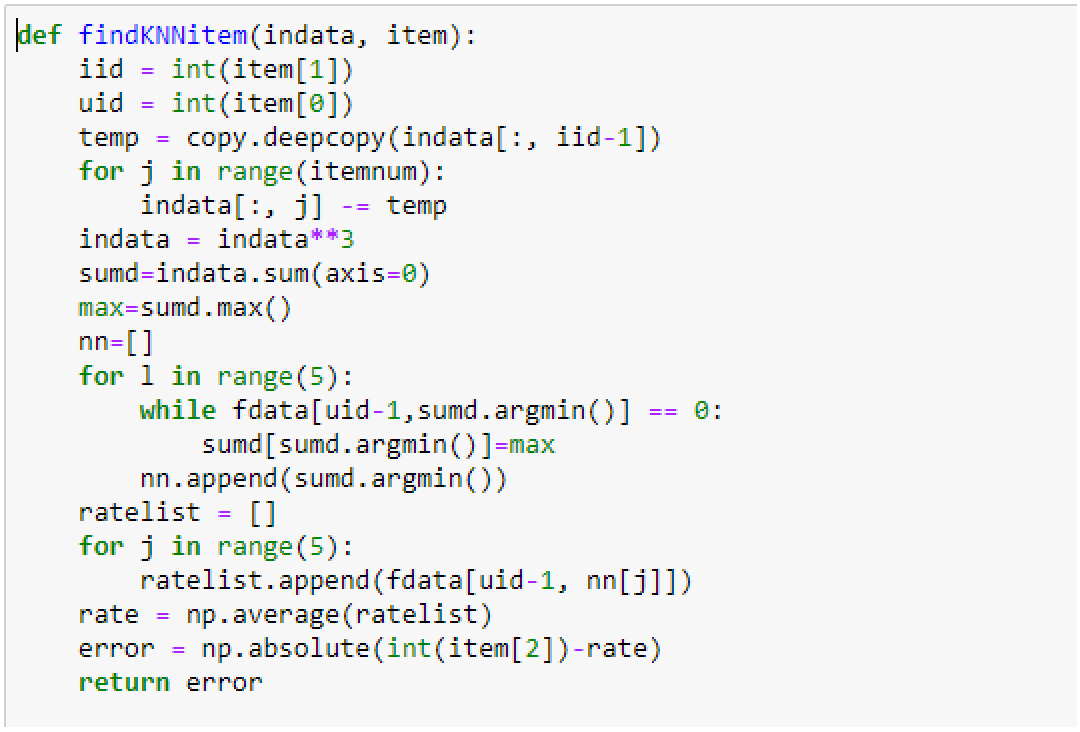
```
print('Formating matrix.')
for row in mdata:
    fdata[row[0]-1, row[1]-1] = row[2]
for case in testset:
    fdata[int(case[0])-1,int(case[1])-1]=0
fdata
```

```
Formating matrix.

array([[5., 3., 4., ..., 0., 0., 0.],
       [4., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [5., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 5., 0., ..., 0., 0., 0.]])
```

Figure 16: formatting matrix

- Next we will define a two functions which are :

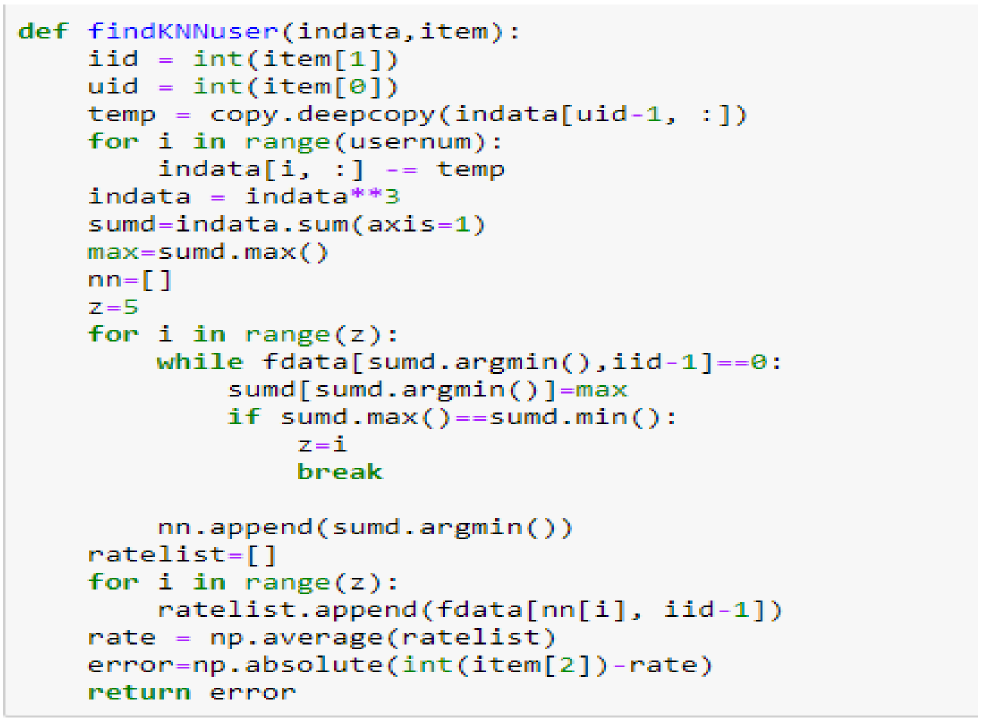
  FindKNNitem :



```
def findKNNitem(indata, item):
    iid = int(item[1])
    uid = int(item[0])
    temp = copy.deepcopy(indata[:, iid-1])
    for j in range(itemnum):
        indata[:, j] -= temp
    indata = indata**3
    sumd=indata.sum(axis=0)
    max=sumd.max()
    nn=[]
    for l in range(5):
        while fdata[uid-1,sumd.argmin()] == 0:
            sumd[sumd.argmin()]=max
        nn.append(sumd.argmin())
    ratelist = []
    for j in range(5):
        ratelist.append(fdata[uid-1, nn[j]])
    rate = np.average(ratelist)
    error = np.absolute(int(item[2])-rate)
    return error
```

Figure 17: FindKNNitem Function

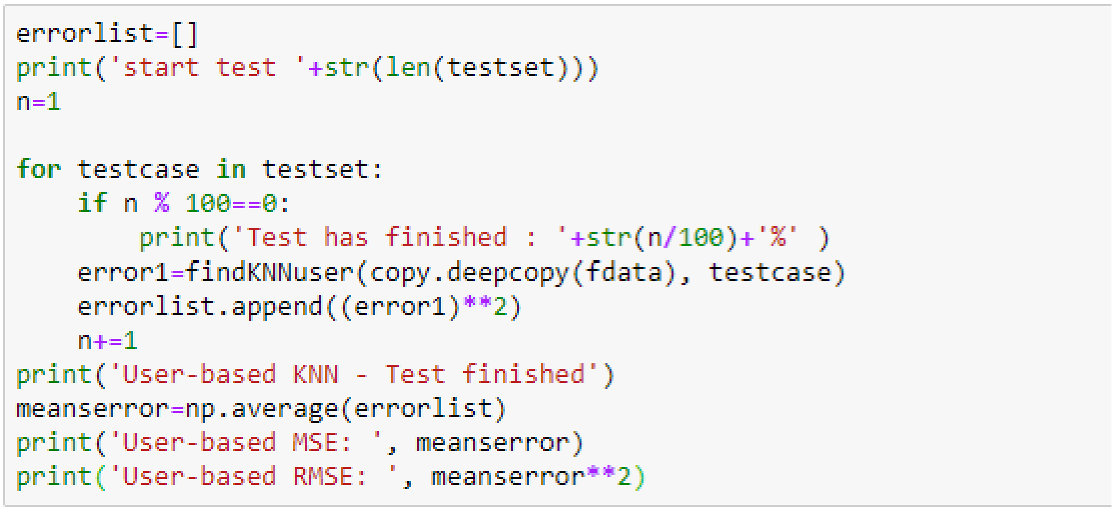
FindKNNuser



```
def findKNNuser(indata,item):
    iid = int(item[1])
    uid = int(item[0])
    temp = copy.deepcopy(indata[uid-1, :])
    for i in range(usernum):
        indata[i, :] -= temp
    indata = indata**3
    sumd=indata.sum(axis=1)
    max=sumd.max()
    nn=[]
    z=5
    for i in range(z):
        while fdata[sumd.argmin(),iid-1]==0:
            sumd[sumd.argmin()]=max
            if sumd.max()==sumd.min():
                z=i
                break

        nn.append(sumd.argmin())
    ratelist=[]
    for i in range(z):
        ratelist.append(fdata[nn[i], iid-1])
    rate = np.average(ratelist)
    error=np.absolute(int(item[2])-rate)
    return error
```

Figure 18: FindKNNuser Function

- Here with the help of the deepcopy and the FindKNNuser function we calculate the RMSE for User-based KNN Approach

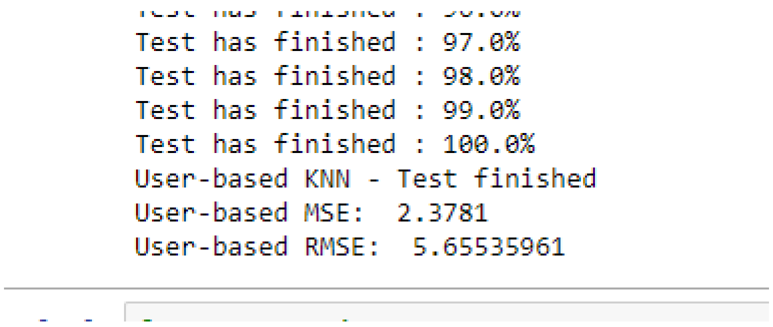


```
errorlist=[]
print('start test '+str(len(testset)))
n=1

for testcase in testset:
    if n % 100==0:
        print('Test has finished : '+str(n/100)+'%' )
    error1=findKNNuser(copy.deepcopy(fdata), testcase)
    errorlist.append((error1)**2)
    n+=1
print('User-based KNN - Test finished')
meanserror=np.average(errorlist)
print('User-based MSE: ', meanserror)
print('User-based RMSE: ', meanserror**2)
```

Figure 19: calculate the RMSE (User-Based KNN Approach)

A deep copy makes a new and separate copy of an *entire* object or list with its own unique memory address. What this means is that any changes you make in the new copy of the object/list won't reflect in the original one. This process happens by first creating a new list or object, followed by recursively copying the elements from the original one to the new one[22].

The result of this code is below:

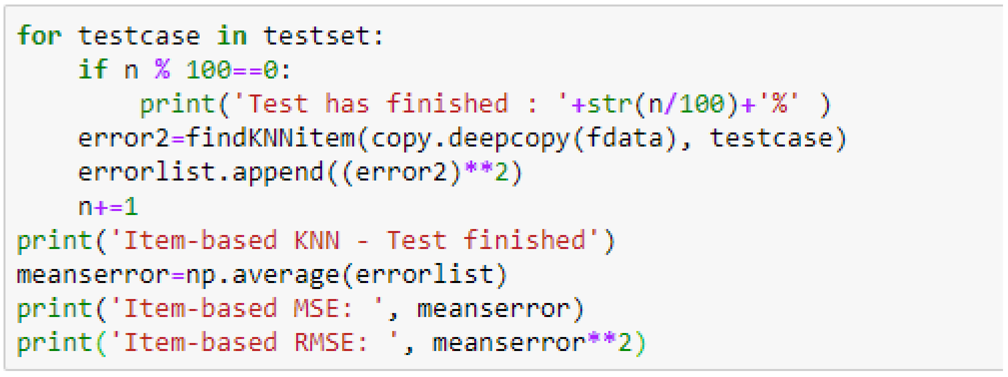


```
Test has finished : 97.0%
Test has finished : 98.0%
Test has finished : 99.0%
Test has finished : 100.0%
User-based KNN - Test finished
User-based MSE:  2.3781
User-based RMSE:   5.65535961
```

Figure 20: User-Based KNN Results

RMSE (UBK) = 5.65

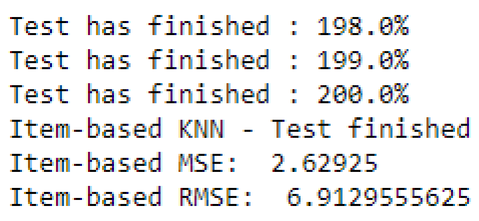
- Next we calculate the RMSE for Item-Based KNN Approach by the FindKNNitem function



```
for testcase in testset:
    if n % 100==0:
        print('Test has finished : '+str(n/100)+'%' )
    error2=findKNNitem(copy.deepcopy(fdata), testcase)
    errorlist.append((error2)**2)
    n+=1
print('Item-based KNN - Test finished')
meanserror=np.average(errorlist)
print('Item-based MSE: ', meanserror)
print('Item-based RMSE: ', meanserror**2)
```

Figure 21: calculate the RMSE (Item-Based KNN Approach)

The result of this code is below:

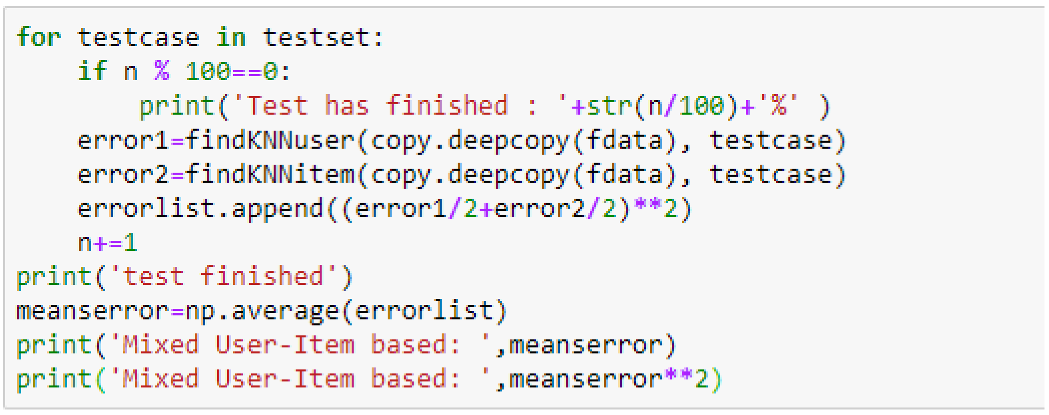


```
Test has finished : 198.0%
Test has finished : 199.0%
Test has finished : 200.0%
Item-based KNN - Test finished
Item-based MSE:  2.62925
Item-based RMSE:  6.9129555625
```

Figure 22: Item-Based KNN Results

RMSE (IBK) = 6.19

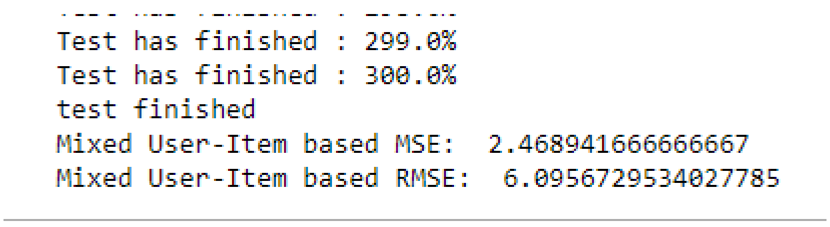
After that we try to mix the two approach together and calculate the RMSE for the User-Item-Based KNN Approach



```
for testcase in testset:
    if n % 100==0:
        print('Test has finished : '+str(n/100)+'%' )
    error1=findKNNuser(copy.deepcopy(fdata), testcase)
    error2=findKNNitem(copy.deepcopy(fdata), testcase)
    errorlist.append((error1/2+error2/2)**2)
    n+=1
print('test finished')
meanserror=np.average(errorlist)
print('Mixed User-Item based: ',meanserror)
print('Mixed User-Item based: ',meanserror**2)
```

Figure 23: RMSE for Mixed User-Item Based KNN Approach

The result of this code is below:



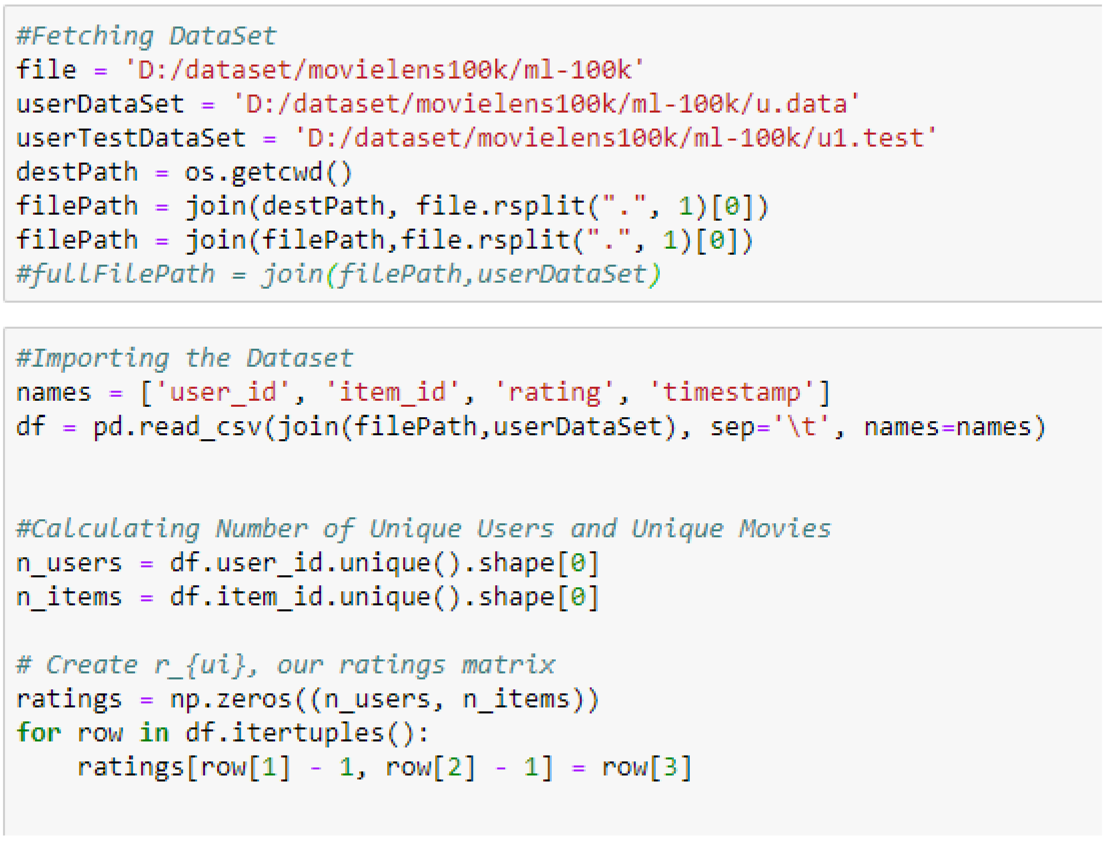
```
Test has finished : 299.0%
Test has finished : 300.0%
test finished
Mixed User-Item based MSE:  2.468941666666667
Mixed User-Item based RMSE:  6.095672953402778 5
```

Figure 24: User-Item Based KNN Results

RMSE (UIBK) = 6.09

3.5. Implementation of Matrix Factorization Algorithm

- Like we do in the KNN first we import libraries and our dataset

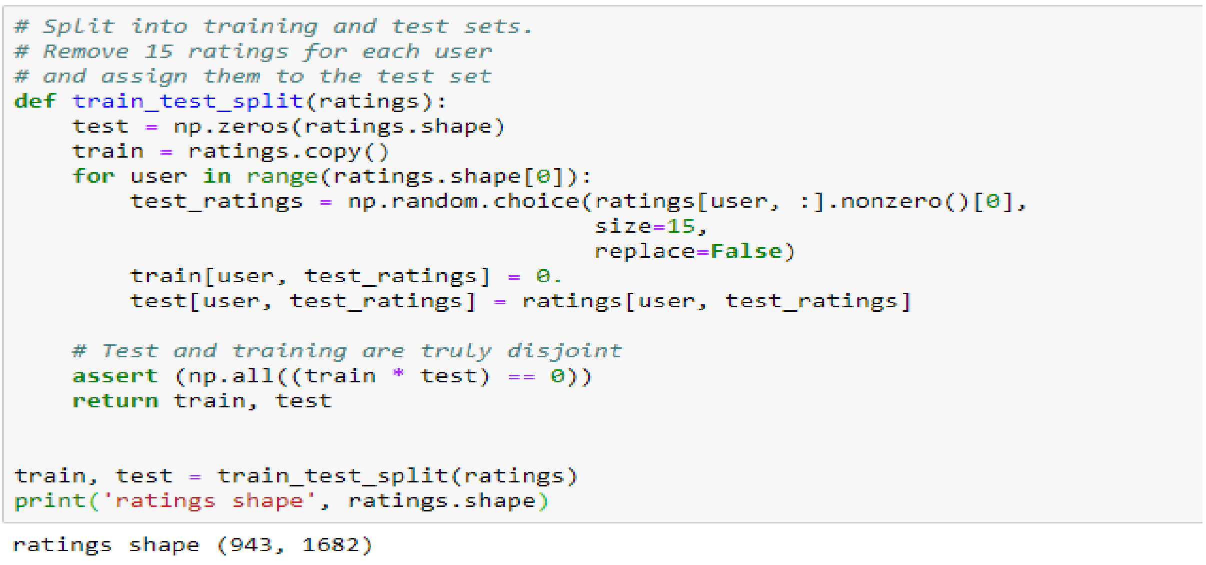


```
#Fetching DataSet
file = 'D:/dataset/movielens100k/ml-100k'
userDataSet = 'D:/dataset/movielens100k/ml-100k/u.data'
userTestDataSet = 'D:/dataset/movielens100k/ml-100k/u1.test'
destPath = os.getcwd()
filePath = join(destPath, file.rsplit(".", 1)[0])
filePath = join(filePath,file.rsplit(".", 1)[0])
#fullFilePath = join(filePath,userDataSet)
```

```
#Importing the Dataset
names = ['user_id', 'item_id', 'rating', 'timestamp']
df = pd.read_csv(join(filePath,userDataSet), sep='\t', names=names)


#Calculating Number of Unique Users and Unique Movies
n_users = df.user_id.unique().shape[0]
n_items = df.item_id.unique().shape[0]

# Create r_{ui}, our ratings matrix
ratings = np.zeros((n_users, n_items))
for row in df.itertuples():
    ratings[row[1] - 1, row[2] - 1] = row[3]
```

Figure 25: Importing libraries and our dataset

- Next we split our data into training and test data

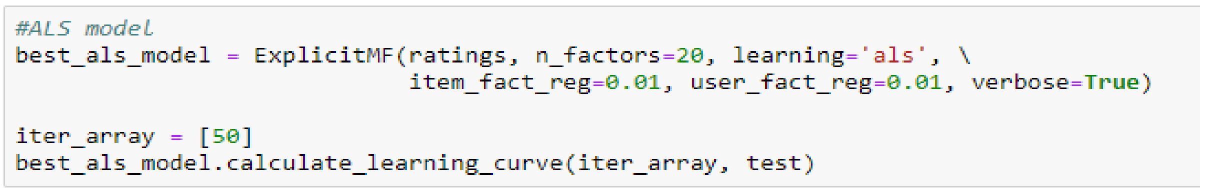


```
# Split into training and test sets.
# Remove 15 ratings for each user
# and assign them to the test set
def train_test_split(ratings):
    test = np.zeros(ratings.shape)
    train = ratings.copy()
    for user in range(ratings.shape[0]):
        test_ratings = np.random.choice(ratings[user, :].nonzero()[0],
                                        size=15,
                                        replace=False)
        train[user, test_ratings] = 0.
        test[user, test_ratings] = ratings[user, test_ratings]

    # Test and training are truly disjoint
    assert (np.all((train * test) == 0))
    return train, test


train, test = train_test_split(ratings)
print('ratings shape', ratings.shape)
```

```
ratings shape (943, 1682)
```

Figure 26: Split dataset into training and test sets

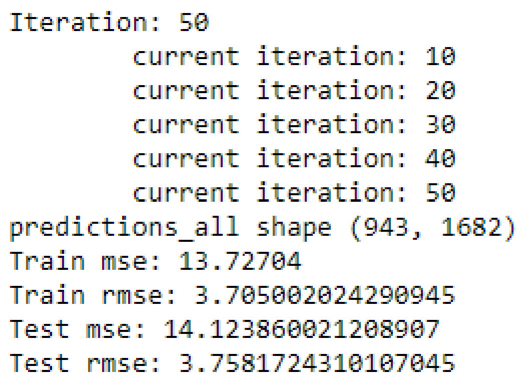
And after defining many function like sgd() and als() we can calculate the RMSE for the Matrix Factorization with Alternating Least Squares (ALS)



```
#ALS model
best_als_model = ExplicitMF(ratings, n_factors=20, learning='als', \
                            item_fact_reg=0.01, user_fact_reg=0.01, verbose=True)

iter_array = [50]
best_als_model.calculate_learning_curve(iter_array, test)
```

Figure 27: Matrix Factorization with ALS Model

The result of this code is below:

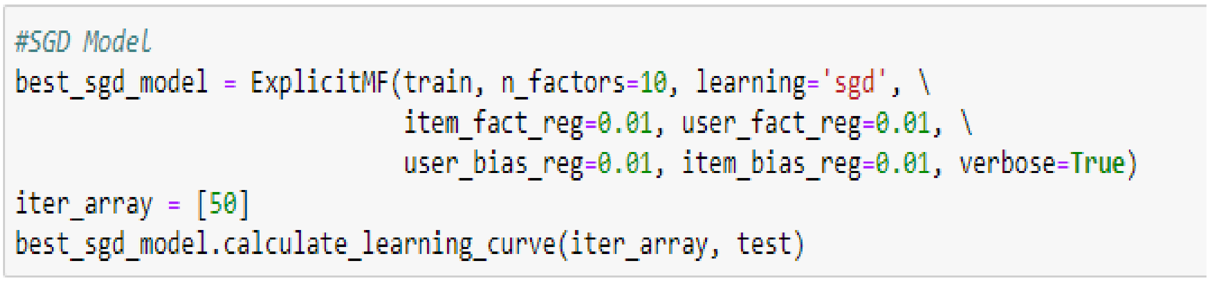


```
Iteration: 50
        current iteration: 10
        current iteration: 20
        current iteration: 30
        current iteration: 40
        current iteration: 50
predictions_all shape (943, 1682)
Train mse: 13.72704
Train rmse: 3.705002024290945
Test mse: 14.123860021208907
Test rmse: 3.7581724310107045
```

Figure 28: Matrix Factorization with ALS Results

RMSE (MF-ALS) = 3.75

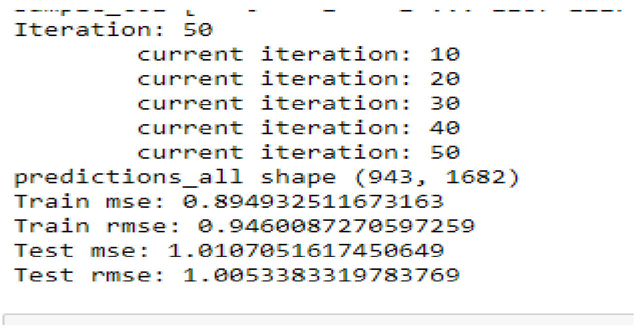
- Calculate the RMSE for the Matrix Factorization with the Stochastic Gradient Descent SGD



```
#SGD Model
best_sgd_model = ExplicitMF(train, n_factors=10, learning='sgd', \
                            item_fact_reg=0.01, user_fact_reg=0.01, \
                            user_bias_reg=0.01, item_bias_reg=0.01, verbose=True)
iter_array = [50]
best_sgd_model.calculate_learning_curve(iter_array, test)
```

Figure 29: Matrix Factorization with SGD Model

The result of this code is below:



```
[illegible]
Iteration: 50
        current iteration: 10
        current iteration: 20
        current iteration: 30
        current iteration: 40
        current iteration: 50
predictions_all shape (943, 1682)
Train mse: 0.894932511673163
Train rmse: 0.9460087270597259
Test mse: 1.0107051617450649
Test rmse: 1.0053383319783769
```

Figure 30: Matrix Factorization with SGD Results

RMSE (MF-SGD) = 1.00

- **Model Comparison**

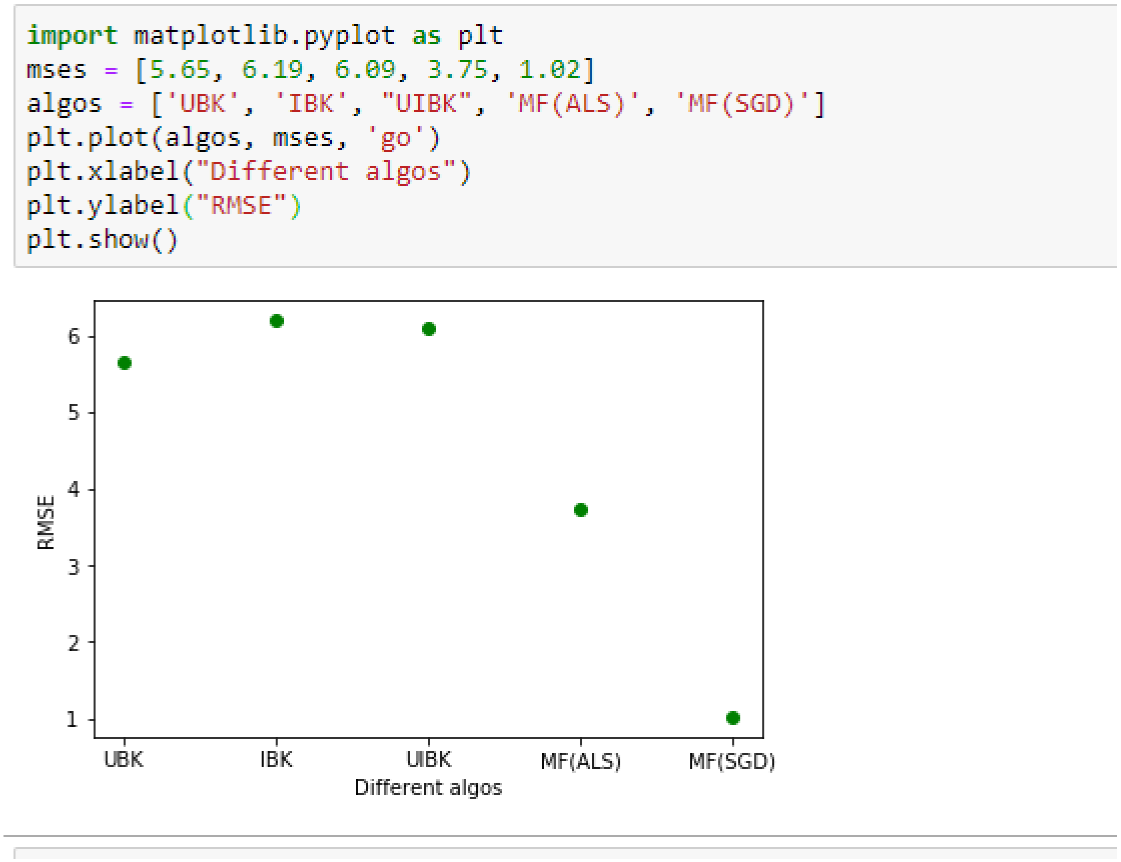
As we mentioned above that we compare between the different algorithms by comparing the values of the Root Mean Square Error which whenever it is close to 0, it indicates the effectiveness of the algorithm.

So we put all the values of the Results (**RMSE**) of such algorithm in Table 1 and we compare between them:

Model	RMSE-Test Set
User-based KNN	5.65
Item-based KNN	6.19
Mixed User-Item Based KNN	6.09
Latent Factor Model (ALS)	3.75
Latent Factor Model (SGD)	1.00

Table 1: the RMSE Values for each models

- By the use of the **matplotlib.pyplot** library we can see the Results below :



```
import matplotlib.pyplot as plt
mses = [5.65, 6.19, 6.09, 3.75, 1.02]
algos = ['UBK', 'IBK', "UIBK", 'MF(ALS)', 'MF(SGD)']
plt.plot(algos, mses, 'go')
plt.xlabel("Different algos")
plt.ylabel("RMSE")
plt.show()
```

Figure 31: RMSE Results in different algorithms

The results in the above table and Figure 34 indicate that the latent factor model with the SGD optimization method performs best among the five variations because it gives us a very small value close to 0 in comparison with other Algorithms. Based on the results, we conclude that the latent factor model with SGD tends to perform better than the neighborhood approaches on the movie rating prediction tasks, provided there is enough data for each user and for each movie.

3.6. Conclusion

In this chapter we implement the two algorithm which are: KNN and MF with the help of machine learning concept and we calculate the RMSE of each Algorithm and compare between it and we find that the RMSE (KNN) **>** RMSE (MF) which inform us that the Algorithm of MF is very good the KNN Algorithm in the prediction of messing value.

General Conclusion

In this thesis, we addressed to the sparse matrix problem in collaborative filtering. This problem happens when the matrix contains a few known data and there are many algorithms try to solve this problem, we choose two Different algorithms witch are the state-of-the-art in the collaborative filtering witch are: KNN and MF for predicting the messing values in the User-item Matrix to give recommendation to the end-user and according to the RMSE value we see that the MF with SGD is the best algorithm to predict missing value.

In our next work we will try to develop an algorithm where the RMSE value is less than 1 when we applicate it to the dataset.

Bibliography

[1] BENATHMANE,L.,DAHMANI,Y.,KHAROUBI,S.,Optimisation sementique d'un systeme de fitrage collaborative 2013 Universite Ibn Khaldoune MAGISTER

[2] [An, 2006] :T.An . COCoFil2 : « Un nouveau système de filtrage collaboratif basé sur le modèle des espaces de communautés » ,2006 .

[3] Jannach, D., Zanker, M., Felfernig, A., and Friedrich, G. (2010). Recommender systems: an introduction. Cambridge University Press.

[4] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, 1998.

[5] P. Resnick, N. Iaocvou, M. Suchak, P. Bergstrom, and J. Riedl. Grouplens: An open architecture for collaborative filtering for netnews. In Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work, pages 175–186, 1994.

[6] S. Deerwester, S. T. Dumais, and R. Harshman. Indexing by latent semantic analysis. Journal of the Society for Information Science, 41(6):391–407, 1990.

[7] T. Hofmann. Probabilistic latent semantic analysis. In Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence, 1999.

[8] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In Proceedings of the 10th International Conference on World Wide Web, 2001.

[9] Aggarwal, C. C. (2016). Recommender systems. Springer.

[10] P. Melville, R. J. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In Proceedings of the 18th National Conference on Artificial Intelligence, 2002.

[11] V. Vapnik, "The Nature of Statistical Learning Theory". Springer Verlag, New York, USA, 1995.

[12] Koren, Y. (2010b). Factor in the neighbors: Scalable and accurate collaborative filtering. ACM Transactions on Knowledge Discovery from Data (TKDD), 4(1):1.

[13] Mnih, A. and Salakhutdinov, R. R. (2008). Probabilistic matrix factorization. In Advances in neural information processing systems, pages 1257–1264.

[14] Eckart, C. and Young, G. (1936). The approximation of one matrix by another of lower rank. Psychometrika, 1(3):211–218.

[15] T. Hofmann. Latent semantic models for collaborative filtering. ACM Transactions on Information Systems, 22(1):89–115, 2004.

[16] Latent Factor Models for Web Recommender Systems.

[17] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. Computer, August 2009

[18] C. Zeng, C.-X. Xing, and L.-Z. Zhou. Similarity measure and instance selection for collaborative filtering. In Proceedings of the 12th International World Wide Web Conference, 2003.

[19] J. Callut, «Implémentation efficace des Support vector Machines pour la classification » Mémoire présenté en vue de l‟obtention du grade de Maître en informatique. Université libre de Bruxelles . département informatique, 2003

[20] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. Computer, August 2009

[21] K. Yu, X. Xu, M. Ester, and H.-P. Kriegel. Selecting relevant instances for efficient and accurate collaborative filtering. In Proceedings of the 10th International Conference on Information and Knowledge Management, 2001.

[22] P.Vincent, « Modèles à noyaux à structure locale », Thèse de Phd en informatique , Université de Montréal,2003

[23] A. Cornuéjols, L. Miclet, Y.Kodratoff, « Apprentissage Artificiel, Concepts et algorithmes » ISBN 2-212-11020-0 , 2002.

[24] M. Rosenstein and C. Lochbaum. What is actually taking place on web sites: Ecommerce lessons from web server logs. In Proceedings of ACM 2000 Conference on Electronic Commerce, 2000.

[25] G.Linden, S.Brent, J.York.«Amazon.com recommendations:Item-to-item collaborative filtering», IEEE internet computing, vol. 7, n°1, p. 76-80, 2003.

[26] N.Denos ,C.Berrut ,L. Gallardo-Lopez ,A. Nguyen . « COCoFil : Une plateforme de filtrage collaboratif orientée vers la communauté », Actes de la 1ère Conférence en Recherche d'Information et Applications (CORIA'04), Toulouse, France, p. 9-26 ,2004.

[27] Resnick, P., Iacovou, N., Sushak, M., Bergstrom, P., and Riedl, J. GroupLens: An open architecture for collaborative filtering of netnews. In Proceedings of the 1994 Computer Supported Cooperative Work Conference. (1994) ACM, New York.

[28] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. Journal of the American society for information science, 41(6):391.

[29] Jannach, D., Zanker, M., Felfernig, A., and Friedrich, G. (2010). Recommender systems: an introduction. Cambridge University Press.

[30] D. M. Chickering, D. Heckerman, and C. Meek. A bayesian approach to learning bayesian networks with local structure. In Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence, 1997.