Product Recommendation System

Paraskevopoulos Athanasios
SID: 3306160004

SCHOOL OF SCIENCE & TECHNOLOGY
A thesis submitted for the degree of
Master of Science (MSc) in Mobile and Web Computing

DECEMBER 2017
THESSALONIKI – GREECE
Product Recommendation System

Paraskevopoulos Athanasios
SID: 3306160004

Supervisor: Assistant Professor Dr. Christos Tjortjis
Supervising Committee Members: Faculty Member Dr. Christos Berberidis
Academic Associate Dr. Marios Gatzianas
Assistant Professor Dr. Christos Tjortjis

SCHOOL OF SCIENCE & TECHNOLOGY
A thesis submitted for the degree of
Master of Science (MSc) in Mobile and Web Computing

DECEMBER 2017
THESSALONIKI – GREECE
Abstract

This dissertation was written as a part of the MSc in Mobile and Web Computing at the International Hellenic University. The aim of this work is to investigate if user’s demographics data and ratings entropy0 scores can have an impact on addressing the cold start problem in collaborative filtering. We propose four collaborative movie recommender systems that use ask-to-rate techniques by displaying movies to users for rating.

The implementation of the aforementioned systems was done in Python 3.6 programming language, developing four independent scripts that display movies for rating using different ask-to-rate techniques: random choice of movies, demographic based, entropy0 based, mix of demographic and entropy0 based.

In the evaluation we have taken into consideration both the accuracy of the predictions but also the user effort. The results have shown that there is (almost) a tie for the first place between demographic-based and entropy0-based systems both in terms of user preference score but also in terms of user’s effort (entropy0 based system is only marginally better). Furthermore, we can also see that the system with the combination of demographics and entropy0, is slightly better (in terms of user preference score) than the basic (random selection), even if the user effort is much higher. Finally, for a future work we can use a movie-set with newer movies or a completely different dataset with another type of products like electronic devices, books etc. Moreover, a mobile implementation can make recommender systems even more useful and also valuable.

Last but not least, I would like to thank my supervisor Dr. Christos Tjortjis for all his valuable support and guidance that he has given me the past six months.

Student Name: Athanasios Paraskevopoulos
Date: 29/12/2017
Contents

ABSTRACT .................................................................................................................................................. III

CONTENTS .............................................................................................................................................. V

1 INTRODUCTION .................................................................................................................................. 1
  1.1 DEFINITION OF RECOMMENDER SYSTEMS ................................................................................ 1
  1.2 PROBLEM ...................................................................................................................................... 3
  1.3 PURPOSE ...................................................................................................................................... 3
  1.4 SCOPE .......................................................................................................................................... 3
  1.5 STRUCTURE OF DISSERTATION .................................................................................................. 4

2 CORE CONCEPTS OF RECOMMENDER SYSTEMS .......................................................................... 5
  2.1 RECOMMENDER SYSTEMS .......................................................................................................... 5
    2.1.1 Basic concepts for ratings ...................................................................................................... 7
  2.2 CONTENT-BASED FILTERING ...................................................................................................... 9
  2.3 COLLABORATIVE FILTERING .................................................................................................... 11
    2.3.1 Item – Based Collaborative Filtering (IBCF) ......................................................................... 11
    2.3.2 User – Based Collaborative Filtering (UBCF) .................................................................... 15
    2.3.3 kNN Algorithm ..................................................................................................................... 16
  2.4 MATRIX FACTORIZATION ........................................................................................................... 18
  2.5 HYBRID ALGORITHM .................................................................................................................. 19

3 CHALLENGES AND RELATED WORK ............................................................................................... 21
  3.1 COLD START PROBLEM ............................................................................................................... 21
    3.1.1 New user .............................................................................................................................. 22
    3.1.2 New product ......................................................................................................................... 22
  3.2 DOCUMENTS ADDRESSING THE COLD START PROBLEM ..................................................... 22
    3.2.1 Using Demographic Information to Reduce the New User Problem in Recommender Systems ................................................................................................................................. 22
    3.2.2 Collaborative Filtering Enhanced By Demographic Correlation ...................................... 24
3.2.3 Cold-start Problem in Collaborative Recommender Systems: Efficient Methods Based on Ask-to-rate Technique........................................... 25
3.2.4 Learning Preferences of New Users in Recommender Systems: An Information Theoretic Approach.................................................. 29

4 REQUIREMENTS AND DESIGN .................................................................. 33

4.1 REQUIREMENTS .................................................................................. 34
  4.1.1 Functional requirements ................................................................. 34
  4.1.2 Non-Functional requirements ......................................................... 35

4.2 SYSTEM ARCHITECTURE .................................................................. 37
  4.2.1 Recommendation engine ............................................................... 38

4.3 USED DATASET ................................................................................... 38

4.4 ASSUMPTIONS .................................................................................... 43

4.5 FLOWCHART DIAGRAMS .................................................................. 43
  4.5.1 Basic algorithm flowchart diagram .................................................. 44
  4.5.2 User Demographics based algorithm flowchart diagram ................. 45
  4.5.3 Movies Entropy0 based algorithm flowchart diagram ....................... 47
  4.5.4 User Demographics and movies entropy0 based algorithm flowchart diagram ................................................................................. 48
  4.5.5 Movies rating function flowchart diagram ........................................ 49
  4.5.6 Inserting target user demographics .................................................. 51

5 IMPLEMENTATION ................................................................................. 53

5.1 BASIC PARTS OF OUR IMPLEMENTATION ......................................... 53
  5.1.1 Functions of the Basic script ........................................................... 53
  5.1.2 Functions of the Demographic-based script ..................................... 54
  5.1.3 Functions of the Entropy0-based script ............................................. 56
  5.1.4 Functions of the Demographic & Entropy0-based script .................... 57

5.2 BASIC FLOW OF EACH SCRIPT ......................................................... 58
  5.2.1 Basic script .................................................................................... 58
  5.2.2 Demographic-based script .............................................................. 58
  5.2.3 Entropy0-based script ..................................................................... 59
  5.2.4 Demographic & Entropy0-based script ............................................ 59

6 EVALUATION AND FUTURE WORK ...................................................... 61
6.1 RESULTS AND EVALUATION..................................................................................61
6.2 CONCLUSIONS....................................................................................................63
6.3 FUTURE WORK ....................................................................................................64

BIBLIOGRAPHY .......................................................................................................65

APPENDIX ..................................................................................................................67
1 Introduction

Over the last decade, the rapid growth of technology has led to an enormous amount of information enabling even an average internet user to have a wide variety of options. Electronic shops like Amazon or E-bay are offering plentitude of products, online newspapers are publishing numerous articles every day while other websites like Netflix are offering thousands of movies to their subscribers. Statistics have also shown that the number of internet users is growing more and more. More specifically we can see that from 2000 to 2017 there is a growth of 976 % in the global internet usage [1].

Although technology evolution is making a huge progress, there are several disadvantages that should be taken into consideration. Using the search engines in order to find something that I would like seem very generic and often has not the desired results. Internet users are facing the problem of information overloading in which it is seems very challenging to find and process the most suitable information in order to extract meaningful information and knowledge [2]. It is obvious that utilizing and manipulating useful information is proven a very demanding and time-consuming procedure.

In order to address this problem, we can use recommender systems which seem an effective solution in many cases. In other words, recommender systems are capable of processing huge amounts of information in order to help users identify meaningful data and knowledge from a wide range of choices. As a result the main goal of these systems is to do all the “hard” work which in other conditions would be executed by human beings.

1.1 Definition of Recommender Systems

Recommender systems suggest items based on users past behavior, preferences and personal data. Because of the diversity of data, the variety of the information and the wide range of products, recommender systems are very essential in order to provide recommendations for products and other items [3]. In other words, a recommender system is a software that process large amounts of data in order to produce useful information. Movies, news or books which are considered by a user as “interesting” can now be presented by these systems that can be very helpful especially in cases with a wide range of items.
We can define a recommender systems as a system that is capable of gathering, processing and suggesting products that might be interesting for a given user. According to [2] a recommender system is every system that outputs personalized recommendations or has the ability to navigate users in order to find interesting and useful products.

Many companies are currently using these systems for commercial reasons. They provide different types of products such as movies, songs or books which means that recommender systems do not focus on specific type of items but they can be generalized in order to operate on a wide range of products. Below we introduce some of the most famous recommendation engines:

- Netflix which is a service for video rental and streaming, is considered as one of the most well-known paradigms because it helps its viewers to find shows that might have not initially chosen.
- Amazon which is a very popular e-commerce website, suggests items that other users have bought based on the item that you have just purchased.
- LinkedIn which is a social networking website designed for business community, makes recommendations for people that you might know, jobs you may like or companies and groups that are interested in.
- Also Hulu which is a streaming video website uses recommender systems is order to suggest content that many users may find interesting.

Recommendations can be optimized if the system can use two different types of input data: explicit (raw user input) and implicit (user’s behavior) [5]. For example, when a user is rating some movies on Netflix, this means that he offers explicit input to the system. Various online communities like MovieFinder or MovieLens for movies and Pandora or Last.fm for music, are trying to collect user opinions, in order to recommend items based on this knowledge. However, there are many cases where this type of information is not available for the system. Moreover, user experience will become worse in case the system will keep using long and demanding questionnaires in order to extract useful information from user.

In order to provide accurate recommendations, systems should be aware of users past ratings and tastes. By this way, system should be able to suggest products based on what users like in the past or products that other similar (based on ratings) users like. However, there are many situations where this kind of knowledge is not available for the new users.
In this case systems have to face the cold start problem which is considered a very critical problem. In order to overcome this problem, many different approaches have been adopted: for example systems are capable of exploiting user’s demographic data in order to make recommendations for new users based only on this kind of information [6].

1.2 Problem

The main problem of collaborative recommender systems is to make suggestions for new users who have just started to make use of the system. In this problem, which is known as cold start problem, the system has to collect new user information in order to be ready for use by the recently entered user. In case the system does not have sufficient information about new users, it will be very hard to provide accurate predictions.

1.3 Purpose

The main goal of this thesis is to address the cold start problem of collaborative recommender systems in the context of suggesting movies to new users. First of all, we will examine how user’s demographic data affects their movies preferences and then we are going to study efficient methods such as the entropy of ratings based on ask-to-rate technique. These approaches will be evaluated for their efficiency and accuracy in the MovieLens 100K dataset.

1.4 Scope

The goal of this study is not to improve the accuracy of collaborative or content based filtering as this is not within the scope of this dissertation. In this study, we are going to investigate if demographic data or/and entropy of information can effectively address the cold start problem. Although there is a wide variety of MovieLens datasets with a lot of movies ratings, the majority of them does not include information about demographic data. In our study, we will use the demographic data of MovieLens 100K dataset combined with small research conducted by us.
1.5 Structure of Dissertation

In the first (Introduction) chapter, we give a brief description of the main features of the recommendation systems and the challenges that they have to face right now. Then, we present the purpose and the scope of this dissertation, and finally we provide a brief summary of the next chapters.

In the second chapter, we present the core concepts of our Thesis. In other words we are trying to describe recommender systems and explain how they work. Furthermore we introduce the main categories of recommender systems and also give a description of their characteristics and their functionality.

In the third chapter, we provide a literature review of the main published work regarding the recommendation systems and how they address the cold start problem. In this point, the main goal is to critique the respective literature and to identify the main problematic areas that can be improved. Moreover, we are trying to define the problem we are working on and also to connect our study with previous knowledge and suggest any further research. More specifically, we are trying to focus on the cold start problem which is a big issue for new users in recommender systems. Also we concentrate on the possibility of addressing it by using user metadata such as demographic data and proving that this kind of information can affect personalized recommendations. Furthermore, we try to focus on how ratings entropy and entropy0 can affect the cold start problem.

In the fourth chapter, we provide a section with the most important functional and non-functional requirements of our proposed system. Moreover, we give a description of the design of our proposed recommender system by providing all the related details and components. In addition, we present some diagrams and also the logic behind our proposed system.

In the fifth chapter, we introduce the implementation of our system by presenting the tools and the programming languages that we have used. For the needs of this thesis the programming language we have used is Python 3.6, and the development environment is PyCharm. Additionally, we present the datasets and all the related knowledge that we have extracted from them. In this case, we have used the MovieLens datasets that include user’s demographic data thousands of movie ratings.

In the final chapter, we evaluate our proposed system, compare it with other systems and present the final conclusions of this study. Furthermore, we are also trying to suggest future work related to our system.
2 Core concepts of Recommender Systems

In this chapter, we introduce the basic idea and some of the most important characteristics that are related with recommender systems. Also, we are trying to explain some of core concepts that are adopted in our dissertation.

2.1 Recommender Systems

In previous chapters we introduced the problem of information overload in which users find it difficult to locate the most suitable information at the right time. The set of solutions that have been proposed can be presented in the following figure:

![Diagram](image)

Picture 2.1: The hierarchy of solutions proposed for the information overload problem

As we can see in the above figure, the solutions are located between information filtering and information retrieval. Information retrieval systems ask the users to specify the type of information that is needed. On the other hand, information filtering systems aim to learn the user’s main interests and then filter information taking into consideration users’ profiles.
As we mentioned in the previous chapters, recommender systems are used to suggest what products to buy, what movies to watch or even who should be your friends on the social media. In 1992 Goldberg created the first recommender system in order to address the problem of the problem of the numerous emails which were presented in user’s mailbox. In simple words, this system is a type of collaborative filtering algorithm in which users make reviews for the emails they read [7]. Over the last decade the need for precise and accurate recommender systems is growing more and more, because there are large amounts of data and the demand of personalized recommendations. Universities and companies have developed many techniques because it is proven that recommender systems can be very profitable.

Recommender systems include two main ingredients: the database and the filtering algorithm [8]. All the datasets and the information about users is placed in database. On the other hand, the filtering algorithm is divided into two steps: Firstly, the algorithm calculates the most similar users or items while in the second step the system is trying to make recommendations for the users. All the above can also be seen in the following figure:

![Information filtering in recommender systems](image)

Below we present the different recommendation techniques that have been developed over the past years:

- **Collaborative filtering**: The recommendations are based on the similarity between users and their ratings.
- **Content-based filtering**: The recommendations are based on the similarity between items.
- **Demographic filtering**: In this case, recommendations are based on the user personal information such as gender, age, occupation, location etc.
- **Social filtering**: Recommender systems are based on user’s social networks
- **Hybrid filtering**: Combination of the above approaches
It is also worth mentioning that our implementation is based on collaborative filtering. In the next sections, we will give a brief description of some of the above techniques focusing on the methods that are used in our proposed system.

2.1.1 Basic concepts for ratings

The rating system has a big impact on the design of the recommender algorithms. The ratings are usually indicate how much a user likes or dislikes a specific item. There are rare cases, where ratings can take continuous values, for example in the Jester recommendation engine the rating values range between -10 and 10. On the other hand, in most cases the rating values are in intervals, which means that there is a collection of distinct ordered numbers used to indicate whether the user likes or dislikes the item on hand. For instance, a 5-star rating system can use the set {-2, -1, 0, 1, 2} where a rating of -2 represents extreme dislike while a rating of 2 an extreme like. For other implementations we may have other distinct values such as {0, 1, 2, 3, 4}, having the same logic as above.

The number of the rating values depends on the recommendation system. The most common scenarios is to use a 5-star, a 7-star or even a 10-star system rating. In figure 2.3 we can see an implementation of a 5 star rating system.

![Picture 2.3: The picture shows a 5-star ratings system that is also referred as interval ratings system](image)
Apart from the ratings, the above picture can also show the semantic meaning of the user’s interests. This meaning can be different depending on the system: For example, Netflix uses a 5-star rating system where 4-stars mean that the user “really liked” the movie, while the “middle” 3-star rating means that the user simply “liked” the movie. For that reason, Netflix has two ratings expressing the “dislike”, and three ratings expressing the “like”, which is also referred to as unbalanced rating scale. There are other implementations where the number of ratings is even and the neutral rating is absent, which leads to a forced choice rating system.

In case of our implementation, we have the Movielens Dataset where there are users that give their ratings for different items. Users are people who rate items, while items are the movies. Ratings can be explicit which means that users inserted the ratings by himself or they can be implicit which means that the ratings were estimated based up on the users behavior. In our implementation, we use a five point rating scale with ratings ranging from 1 to 5 (1 is the extreme dislike, while 5 is the extreme like).

In table below we can see an example of the rating matrix that will be used in our implementation:

<table>
<thead>
<tr>
<th></th>
<th>Toy Story</th>
<th>Men in Black</th>
</tr>
</thead>
<tbody>
<tr>
<td>User1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>User2</td>
<td>5</td>
<td>Nan</td>
</tr>
<tr>
<td>User3</td>
<td>Nan</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2.1: Example of rating matrix

The above rating matrix includes the ratings which three users gave for 2 movies. As we have mentioned before the ratings are ranging between 1 and 5. Also, the Nan value means that either the user have not rated the movie because either he has not seen it or he has seen it, he has not managed to rate it.
2.2 Content-Based filtering

Over the past years, a lot of research has been carried out in order to address the problem of information overload, as we have mentioned above. Several items are compared with items that have already rated by users in order to recommend the most suitable items.

The difference between information filtering and information retrieval can be located in this point: The user is not trying to make a query for information, but the filtering system is building a model based on user’s past choices and behavior and then tries to recommend the most suitable information to the user. Although there is a difference between these two concepts, information filtering has adopted several approaches from information retrieval, as it can be seen in content-based filtering and in collaborative filtering.

In content-based filtering, the recommendation is constructed based up on user’s behavior. In this technique which is also known as cognitive filtering [10], all the available information is used in order to predict its relevance taking into consideration the user’s profile. Content-based filtering has many similarities with the relevance feedback of information retrieval literature [11] in which the query vector is constructed by using the relevance of user’s opinions on new documents. In Information Filtering (IF), this modified query vector can be considered as a profile model that includes keywords and their relative importance. Based on this profile, the relevance of new items is calculated by measuring the similarity between the query vector and the item feature vector.

In their simplest form, these profiles are user defined keywords or rules that represent users interests and traits. Usually, users would prefer the system to learn their profiles rather than providing it to system by themselves. For that reason, systems have to use machine learning techniques where the main idea is to learn to create rules and classify newly entered items based up on previous knowledge that has been provided by users. Thus, machine learning techniques can be used in order to construct a model that will be capable of predicting whether newly introduced products or items are probably going to be of interest. The ML techniques used in this case are based on text categorization because the IF is mainly focused on textual domains [19].

The process of allocating a Boolean value to each pair \((d_j,c_i) \in D \times C\), where \(D\) is a set of documents and \(C\) is a set of categories. If true is allocated to the pair \((d_j,c_i)\) there is a
tendency to assign the category $c_i$ to the document $d_j$. On the other hand, if false is assigned to this pair then there is an aversion to assign the category $c_i$ to the document $d_j$. The aim of this process is to estimate the function $F: \mathbb{D} \times \mathbb{C} \rightarrow \{\text{True}, \text{False}\}$ which determines the way that documents should be classified, by defining a new function $F': \mathbb{D} \times \mathbb{C} \rightarrow \{\text{True}, \text{False}\}$. The $F'$ function which is called model, should be similar to $F$ as much as possible.

One of the most well-known approaches from the Information Retrieval and Text Categorization domains is the Rocchio Algorithm which describes documents using the vector space representation having as the most important ingredient the TF-IDF weight (Term Frequency/Inverse Document Frequency). TF-IDF can be formulated by the following equation:

$$
tf idf(t_k, d_i) = tf(t_k, d_i) \times \log \frac{N}{n_k} \quad (1)
$$

where $N$ represents how many documents there are in the collection, and $n_k$ defines the number of documents that involve the token $t_k$. Moreover, $tf(t_k, d_i)$ can be considered as the scheme which calculates how many times the token $t_k$ appears in document $d_i$.

Rocchio algorithm calculates $\vec{c}_i = (\omega_{1i}, \ldots, \omega_{T|i})$ (where $T$ is the number of the unique tokens in the training set), for the $c_i$ category using the following equation:

$$
\omega_{ki} = \beta \cdot \sum_{d_j \in [\text{POS}]_i} \frac{\omega_{kj}}{\text{POS}_i} - \gamma \cdot \sum_{d_j \in [\text{NEG}]_i} \frac{\omega_{kj}}{\text{NEG}_i} \quad (2)
$$

where $\omega_{kj}$ represents the tfidf factor of the token $t_k$ in the $d_j$ document, $[\text{POS}]_i$ is the positive example in the training set for a category $c_i$ and $[\text{NEG}]_i$ is the negative example respectively.

Moreover $\beta$ and $\gamma$ are the factors that set the relative weight of the positive and negative examples. The vector model enables us to estimate how similar two vectors are taking into consideration the correlation. In order to calculate this correlation we can simply measure the cosine of the angle between these vectors. The class $\vec{c}$ is assigned to a document $d_j$, by computing the similarity between each vector $\vec{c}_i$ and the document $\vec{d}_j$. The $\vec{c}$ will eventually be defined by the $c_i$ that has the highest similarity score.
2.3 Collaborative filtering

In Collaborative filtering (CF), user similarity is calculated based on user’s ratings [9]. The majority of the studies uses this technique because it is easy to implement and also it provides satisfactory results. Most of the research is mainly focused on two points: The first is how to determine the similarity metric and the second is how to make predictions for items that have no ratings.

Collaborative filtering is mainly based on a model that exploits user’s behavior. This model can be created by using a single user’s behavior or by using the behavior of a group of users that have similar preferences. In other words, this technique makes recommendations based up on the collaboration of many users and focuses on users that have similar traits and behavior.

There are two main groups of Collaborative filtering algorithms: Memory-based Collaborative Filtering and Model-Based collaborative Filtering [12]. In Memory-Based Collaborative Filtering, recommendations are made by using the whole or a large part of the user dataset. One of the most famous and effective algorithms of this category is User-Based Collaborative Filtering. One of the most important disadvantages of this algorithm is that in order to create recommendations, the system has to process the whole dataset which can be proved a very demanding and slow task. On the other hand, model-based algorithms utilize datasets in order to create a more concrete model that will be used in order to make recommendations. The most famous algorithm in this case is Item-Based Collaborative Filtering. Below we are going to introduce the core concepts of the most famous memory and model based collaborative filtering algorithms. Furthermore, we will introduce kNN algorithm, a well-known data science algorithm that will be used in combination with Collaborative Filtering in our implementation.

2.3.1 Item – Based Collaborative Filtering (IBCF)

Item Based collaborative filtering [13, 14] is a model-based technique which recommends items taking into consideration the relationship of items from the rating matrix. The main idea of this technique is that users would choose items which have similarities with other items they had already liked in the past.

This approach involves the calculation of similarity matrix which has item to item similarities based on a similarity measure. Some of the most well-known similarity measures are: Cosine Similarity, Pearson Similarity and Adjusted Cosine Similarity.
These similarities are located in a matrix $S$ with $n$ rows and $n$ columns. In order to the size of the matrix to $n \times k$ where $k \ll n$, we only save the $k$ most similar items and their corresponding similarity values. The neighborhood of the item $i$ with size $k$, can be defined by the set $S(i)$ which is translated as the $k$ items that are most similar with the item $i$. Keeping the $k$ most common neighbors simplifies the problem, however it is obvious that the quality of recommendation is decreasing.

One of the most important parts in item-based recommender systems is to calculate item similarities and then to choose only the most similar items. The main idea behind the calculation of the similarity between two items $i$ and $j$ is to choose only the users who have rated both of these items and then to calculate the similarity $s_{ij}$ based on a similarity measure technique. The above procedure can be visualized in the following Picture 2.4, where there is a matrix with $m$ rows which represent users and $n$ columns that represent items.

![Picture 2.4: Elicitation of the co-rated items and similarity computation.](image)

There three main methods that are used in order to calculate the similarity between items. These methods which are described below are: cosine based similarity, correlation based similarity, and adjusted cosine similarity.
2.3.1.a. Cosine-based similarity

In this approach, two items are represented as two vectors in the m dimensional user space. We calculate the similarity between these items by measuring the cosine angle of their corresponding vectors. Looking at the matrix of Picture 2.3 we calculate the similarity between two items i,j (sim(i,j)):

\[
sim(i, j) = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{|\vec{i}| * |\vec{j}|}
\]

(3)

where “\cdot” is the dot product

2.3.1.b. Correlation-based similarity

In this technique, we calculate similarity between two items i,j by computing the Pearson correlation corr_{ij}. In order to improve accuracy we have to single out the cases where users have rated both items i and j as we have mentioned in Picture 2.3. Defining as U the set of users that have rated both items i and j, correlation-based similarity is calculated by the following equation:

\[
sim(i, j) = \frac{\sum_{u \in U}(R_{ui} - \bar{R}_i)(R_{uj} - \bar{R}_j)}{\sqrt{\sum_{u \in U}(R_{ui} - \bar{R}_i)^2} \sqrt{\sum_{u \in U}(R_{uj} - \bar{R}_j)^2}}
\]

(4)

where \(R_{ui}\) corresponds to the rating of the user u for the item i, and \(\bar{R}_i\) represents the average rating for the i-th item.

2.3.1.c. Adjusted cosine similarity

The calculation of similarity between user-based collaborative filtering and item-based collaborative filtering has some differences. One of the main differences is that in user based collaborative filtering, we calculate the similarity between the rows of the rating matrix while in item based collaborative filtering we calculate the similarity between columns of the matrix (each pair of the co-rated items represents a different user). In case of item based collaborative filtering, the computation of similarity by the cosine similarity technique has one main disadvantage: The variation of the rating scale for different users is not taken into consideration. This problem can be addressed by using the adjusted cosine similarity where the user average is subtracted by each co-rated pair.

By using this approach, the similarity between two items i and j is calculated by the equation:
\[
\text{sim}(i, j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R}_u)(R_{u,j} - \bar{R}_u)}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R}_u)^2 \sum_{u \in U}(R_{u,j} - \bar{R}_u)^2}} \tag{5}
\]

where \( R_u \) denotes the average rating for user \( u \).

### 2.3.1.d. Prediction computation

The most crucial part of a collaborative filtering system is to generate the desired predictions. First of all we try to single out the most similar items taking into consideration the aforementioned similarity measures and then we try to focus on the users’ ratings and generate predictions based on the weighted sum approach.

In this approach we calculate the prediction for an item \( i \) targeting a user \( u \) by calculating the sum of all the ratings the user \( u \) has given for items that are similar to the item \( j \). The similarity \( \text{sim}(i, j) \) between items \( I \) and \( j \) “weights” each one of the ratings. Taking into consideration the idea that is depicted in Figure 2.5 we can define the prediction \( P(u,i) \) using the following equation:

\[
P_{u,i} = \frac{\sum_{all\ similar\ items,N}(s_{i,N} \times R_{u,N})}{\sum_{all\ similar\ items,N}(|s_{i,N}|)} \tag{6}
\]

The main idea of this technique is that it tries to describe how a user rates similar items. Moreover we can see that the sum of similar items scales the weighted sum in order to guarantee that the prediction falls within a specified range.

---

*Picture 2.5: Item-based collaborative filtering algorithm. Prediction is done by considering 5 neighbors*
2.3.2 User – Based Collaborative Filtering (UBCF)

User based collaborative filtering is memory based approach that uses and process rating data from a wide range of users. The main idea behind this technique is that users with the same or almost the same preferences will also give similar ratings in the items. For that reason, a user’s missing ratings can be estimated by finding the k-most common neighbors and then process their ratings in order to create the final recommendations.

The neighborhood with the k most common users is calculated by the similarity of users, choosing the k most similar users or choosing all users that have a defined similarity threshold. For User-based collaborative filtering the most well-known similarity measures are person correlation and cosine similarity (both of them where presented in the previous paragraph 2.3.1). As we have mentioned before, a user’s N(a) ⊂ U neighborhood can be estimated either by choosing a number of the most similar neighbors or by defining a threshold on the similarity. After finding the common neighbors, we aggregate their ratings in order to predict the missing ratings for the target user uₐ. A simple approach is to take the average of the ratings in the neighborhood by using the below equation:

\[ \hat{r}_{aj} = \frac{1}{N(a)} \sum_{i \in N(a)} r_{ij} \quad (7) \]

In order to exploit the case the fact that some neighbors are more similar to the target user than other neighbors, we adopt another approach in which we add weights in equation (7). As a result we now have:
\[ \hat{r}_{aj} = \frac{1}{\sum_{i \in N(a)} s_{ai}} \sum_{i \in N(a)} s_{ai} r_{ij} \]  

(8)

where \( s_{ai} \) defines the similarity between a neighbor \( u_i \) and the target user \( u_a \).

An even better approach is to measure \( \hat{r}_{aj} \) with distinct ratings. In this case, we have the classic user-based collaborative filtering algorithm, in which there is an aggregation of the target user’s average rating, considering all the items that the target user has rated. This approach can be defined by the following equation:

\[ \hat{r}_{aj} = \bar{r}_a + \frac{\sum_{u=1}^k (r_{u,j} - \bar{r}_u) p_{a,u}}{\sum_{u=1}^k p_{a,u}} \]  

(9)

where: \( k \) is the number of the \( k \) most similar neighbors for the target user \( a \), \( p_{a,u} \) denotes the similarity between the target user and the other users \( u \), \( \bar{r}_a \) is the average rating for the target user \( a \), \( \bar{r}_u \) is the average rating of the neighborhood users for the item \( j \), and \( r_{u,j} \) is the rating that user \( u \) gave to item \( j \).

2.3.3 kNN Algorithm

In this point, we will give a short description of the kNN algorithm which is used in our implementation. K Nearest Algorithm (known as kNN) is very simple and easy to understand and also has an incredibly well performance. Moreover, it is versatile and robust classifier and has a wide range of applications. The aim of this algorithm is to utilize a dataset where the data points divided into classes, in order to predict in which class belongs a new data point.

Each of the features of the dataset is considered as a different dimension in space and the value of an observation for each of these features is considered as a coordinate, which means that we have a collection of points in the dimensional space. Eventually, the similarity of two points can be regarded as the distance between them.

In order to make predictions for a new observation the algorithm picks the \( k \) most similar (closest distance) points to this observation and then chooses the most similar class between them. For that reason the algorithm is referred to as k-Nearest Neighbors Algorithm. [23]

kNN is considered as a non-parametric and lazy algorithm. It is characterized as non-parametric because it does not make any assumptions regarding the underlying data dis-
tribution. This is a big advantage because the majority of the practical data does not usually follow the typical theoretical assumptions. As a result, non-parametric algorithms such as kNN are here to offer the desired solutions.

Moreover kNN is a lazy algorithm. In other words, it does not utilize the training data sets in order to make generalizations. This means that the explicit training phase is absent or is extremely small, which denotes that the training phase is very fast. The absence of generalization indicates that kNN maintains all training data which is needed for the testing phase. Contrary to other methods in which you can remove a portion of the dataset, lazy algorithms such as kNN utilizes the whole dataset in order to make predictions.

In case of kNN, two opposing parts can also be observed: Although the training is absent or almost absent, the testing phase is much more expensive regarding memory and time. Time is demanded because there are cases in which all data had to participate in generating a prediction, while memory is needed when all data must be saved.

The algorithm can be briefly described in four steps:
1. We define a positive integer k, and also a new sample
2. We choose the k points from our dataset, which are most similar to the new sample
3. We make the classification based on these points
4. The aforementioned class is given to our new sample

As we have mentioned before, data points are located in a feature space and can be considered as scalars or multidimensional vectors. As a result, there is the concept of distance between these points, which can be calculated by many ways for example the Pearson correlation or simply the Euclidean distance. Furthermore, we take into consideration the integer number k which determines the number of neighbors that defines the classification.

To conclude, considering that the points are m-dimensional the procedure of finding the k-Nearest Neighbors can take O(m) time. Moreover, choosing the value of number k is a challenging task: If the value of number k is small the noise will have a big impact on the final result. On the other hand, if the value of number k is large, then the algorithm becomes more resource demanding. As a result, a compromised solution for this issue is to choose the number k based up on the function \( k = \sqrt{n} \), where \( n \) is the number of data that are included in the dataset.
2.4 Matrix Factorization

Until now we have introduced neighborhood methods which are focused on measuring the relationships between items or users. In contrast to these methods, there are also the latent factor models where both items and users are characterized on factors which are derived from the rating patterns.

The implementation of latent factor models is mainly based on matrix factorization [22]. The basic functionality of the Matrix Factorization is that it characterizes items and users using vectors of factors which are derived from item rating patterns. Recommendations are generated in case there is high similarity between items and users. These approaches are becoming more and more popular in recent years because they combine scalability and precise recommendations. Furthermore, they are also more flexible because they are capable of modeling real life scenarios.

As we have mentioned in the previous section, recommender systems are based on various types of input data. This data is usually located in matrices which have one dimension as the items and the other dimension as the users. One of the most suitable type of data is the explicit feedback, which includes users’ ratings for products in interest. For instance, Netflix gives users the opportunity to rate their preferred movies by giving star ratings, while TiVo collects ratings by enabling users to press thumbs up or thumbs down buttons if they like or not the movies respectively. The explicit feedback can also be called as ratings. It is also worth noting that these ratings are often placed in sparse matrices because users have only rated only a small number of the existing movies of the whole dataset.

One of the main advantages of matrix factorization is that it is capable of enabling the integration of additional information. In case explicit feedback is not available, recommender systems can predict user tastes by using implicit feedback. Implicit feedback represents opinions by taking into consideration users’ behaviors and habits such as: past purchases, search history and in some cases the movement of mouse. Furthermore, implicit feedback can be depicted by a dense matrix because it often indicates if there is an event or not.
2.5 Hybrid Algorithm

Hybrid techniques combine multiple recommendation algorithms (e.g. content-based, collaborative filtering, etc.) increasing the efficiency and the likelihood to generate more precise recommendations as well as the complexity of recommender systems. In order to combine these methods many approaches have been proposed.

Generated recommendations can be significantly boosted by using a hybrid recommender that utilizes several of the aforementioned methods. A well-known approach is the combination of content-based and collaborative filtering. Hybrid recommender systems may be a smart solution for addressing the cold start problem which is one of the most serious problems of recommender systems. Figure 2.7 illustrates the basic idea of the generated recommendations for a new introduced user based up on a hybrid recommender which combines social links and collaborative filtering.

![Picture 2.7: The main idea behind a hybrid recommender system](image-url)
3 Challenges and related work

Recommender systems have been a subject of a vast number of research studies and discoveries in order to find new approaches capable of enhancing and improving recommendations. In this chapter we are going to present some of the most important research topics over the last years. This analysis is mainly based on papers [15, 16, 17, 18]. Moreover, the majority of the presented work is tightly related with the development and the problems that we have faced in our work. To sum up, we will present the cold start problem, followed by some works that are trying to solve it using various techniques and approaches.

3.1 Cold Start Problem

One of the most challenging problems which recommends systems have to face is undoubtedly the limited number of the initially available user data. Under these circumstances, it is not easy to apply the aforementioned recommendations techniques and especially the collaborative filtering method. Although knowledge based or content based models proved to be more resistant to cold start problems than collaborative filtering, it is not easy to have this knowledge or content always available. Small numbers of data has negative effect on the performance of recommender systems by downgrading their prediction accuracy to a large extent. For that reason, there is a great interest of researching and studying all the drawbacks of limited data, and also what has to be done in order to address this problem.

Cold start problem affects recommender systems in terms of new users and new products [20]. Many studies have shown that cold start problem affects all types of recommender systems, but it is also proved that collaborative filtering methods have to face bigger problems than content based methods. [5] Below we present the two different categories of the cold start problem which are related with the new user and the new product respectively.
3.1.1 New user

The cold start problem is related with the issue of the new user when a new user has just been introduced himself to the system or when an already existing user has not given enough data to the system and as a result the system is not performing with the normal way. Having this limitation in user’s data, the system generates inaccurate predictions which do not fit with the users preferences. In order to address this problem, many systems use the ask-to-rate technique where they ask from users to rate some products. For example MovieLens asks users to rate movies when they sign up [21].

3.1.2 New product

The second instance of the cold start problem is related with the issue of the new product. This problem arises when a new product is introduced to the system because is not related with any user or any of the already existing products. This limitation in data is very challenging especially in case of collaborative filtering systems which usually use information that describe connections between products and users. On the other hand, content based filtering methods are more robust because they classify the items based up on its characteristics. [21]

3.2 Documents addressing the cold start problem

Below we present some solutions that have been proposed in the past, by giving a description of some of the documents which helped us to create our recommender system.

3.2.1 Using Demographic Information to Reduce the New User Problem in Recommender Systems

In [15], the author attempts to build a recommender system based up on the demographics data included in the MovieLens 100K dataset. This dataset which will also be used in our work, has various information including 100,000 movies ratings made by 963 users for 1682 movies. Moreover, the ratings range from 1 –the lowest rating- to 5 – the highest rating. Additionally, the dataset is created by users’ personal information which are were provided when users visited the MovieLens website for the first time. The structure of the demographic information is given below:

user id | age | gender | occupation | zipcode
Furthermore, the user ratings in the dataset is given as follows:

| user id | item id | rating | timestamp |

Dividing the above users into training and testing set, we create a model that has K distinct clusters, and we train it, by taking into consideration the users of the training set. This model can be considered as a classifier that defines which of the users of the training set matches a new introduced user (a user from testing set). Also, this model makes the classifications based on the user’s demographic data. The ratings of the new introduced user are generated by taking into consideration the rating information which is calculated from users who belong to the same cluster.

Some indicative rating predictions for users in the testing set are plotted along with the actual ratings of the users of training set on some specific movies. These plots are shown in picture 3.1:

![Picture 3.1](image.png)

Picture 3.1: This plot depicts the predicted ratings $r^*$ along with the actual ratings $r$, for five movies: Three Colors White (1994), Grand Day Out (1992), A (1992), Gone with the Wind (1939), Glengarry Glen Ross (1992), Angels and Insects (1995). The figure shows that the predicted ratings $r^*$ are very close with the actual predictions $r$. However, regarding the final results it was observed that most of the predicted ratings are located around 3–4.

To sum up, the prediction of ratings for new users assuming that there are not any past rating data, cannot guarantee that there is any significant relevance between the number of clusters and the precision of the prediction regarding the MovieLens 100K
dataset. Moreover, it seems that there is a connection between demographic data and movie ratings, but in order to generate more concrete results there is a need for a greater range of demographic data.

3.2.2 Collaborative Filtering Enhanced By Demographic Correlation

In [16], there is an attempt to introduce a specific technique which includes a lot of approaches of existing algorithms combing them with demographic data using MovieLens 100k dataset. The introduced hybrid algorithms called U-Demog and I-Demog, are mainly influenced by the User-based and Item-based collaborative filtering respectively. Additionally, the aforementioned algorithms are also enhanced by the user’s demographic data: age, gender, a choice of 21 occupations, and also the zip code for each user that gave his ratings. The aforementioned data is used for the calculation of demographic correlations by taking into consideration the user vector similarities. In other words, each user in MovieLens 100k dataset corresponds to a user demographic vector that is defined as a vector with 27 features and can be seen in detail in the following table:

<table>
<thead>
<tr>
<th>feature #</th>
<th>feature contents</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>age &lt;= 18</td>
<td>• each user belongs to a single age group,</td>
</tr>
<tr>
<td>2</td>
<td>18 &lt; age &lt;= 29</td>
<td>• the corresponding slot takes value 1 (true)</td>
</tr>
<tr>
<td>3</td>
<td>29 &lt; age &lt;= 49</td>
<td>• the rest of the features remain 0 (false)</td>
</tr>
<tr>
<td>4</td>
<td>age &gt; 49</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>male</td>
<td>• the slot describing the user gender is 1</td>
</tr>
<tr>
<td>6</td>
<td>female</td>
<td>• the other slot takes a value of 0</td>
</tr>
<tr>
<td>7-27</td>
<td>occupation</td>
<td>• a single slot describing the user occupation is 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• the rest of the slots remain 0</td>
</tr>
</tbody>
</table>

Table 3.2: Description of the user demographic vector
In this point, it is worth mentioning that we will follow the same logic in our implementation where we are also exploiting user’s demographic data. This procedure is described in chapter 4 (requirements and design).

The experiments have shown that the performance of the proposed algorithms can be much better than the base algorithms, but on the other hand it can also be worse than them. This deviation is mainly based on the role of the demographic correlations in the process of the prediction generation.

As it has also mentioned in the previous section, it was noticed that the demographic data from the MovieLens 100K dataset, does not have the adequate information in order to generate precise and reliable predictions. However, in case this data is combined with other types of filtering, like collaborative filtering, the recommendation procedure can be boosted and the final predictions can be more precise and reliable.

### 3.2.3 Cold-start Problem in Collaborative Recommender Systems: Efficient Methods Based on Ask-to-rate Technique

In this document [17], the author is trying to address the cold start problem of a Collaborative filtering recommendation method by proposing some variations of the “ask-to-rate” technique.

In order to generate recommendations, the author uses memory based Collaborative filtering algorithm which was described in the previous Chapter (Chapter 2). In this point we can figure out that this algorithm is mainly based on kNN (k –Nearest Neighbors) algorithm. The whole recommendation procedure can be described as follows:

- First of all, we calculate the similarity between the active user and the other users who have rated the item by measuring the Pearson’s correlation. Defining as U the set of users that have rated both items i and j, correlation-based similarity is calculated by the following equation:

  \[
  \text{sim}(i,j) = \frac{\sum_{u \in U} (R_{u,i} - \bar{R}_i)(R_{u,j} - \bar{R}_j)}{\sqrt{\sum_{u \in U} (R_{u,i} - \bar{R}_i)^2} \sqrt{\sum_{u \in U} (R_{u,j} - \bar{R}_j)^2}}
  \]  

  (10)

  where \( R_{u,i} \) corresponds to the rating of the user u for the item i, and \( \bar{R}_i \) represents the average rating for the i-th item.

- Then, the prediction for a target item regarding an active user can be measured by the following equation:
\[
\hat{r}_{aj} = \bar{r}_a + \frac{\sum_{u=1}^{k}(r_{uj} - \bar{r}_u)P_{a,u}}{\sum_{u=1}^{k} P_{a,u}} 
\]

where: \(k\) is the number of the \(k\) most similar neighbors for the target user \(a\), \(P_{a,u}\) denotes the similarity between the target user and the other users \(u\), \(\bar{r}_a\) is the average rating for the target user \(a\), \(\bar{r}_u\) is the average rating of the neighborhood users for the item \(j\), and \(r_{u,j}\) is the rating that user \(u\) gave to item \(j\).

Some of the advantages of Collaborative filtering algorithms are that they are simple to implement and relatively easy to understand, and also that new data can be added without any problem. However, the main drawback of these systems is the cold start problem when there is a new user to the system.

In order to find a solution for the cold start problem, the author proposes the “ask-to-rate” technique. The main idea of this technique is to present some items to the new user and ask for explicit ratings. Then, in the user item matrix, the row with the ratings of the new user is not empty anymore and the system is capable of using these ratings in order to make recommendations. The above process can also be depicted below:

![Diagram](image)

Picture 3.2: The main idea behind ask-to-rate technique

It is also worth noting that the system must be capable of presenting the most informative items in order to collect the right information for the new user. If the ratings of the new user are originated from a well-designed selection method rather than a “random selection”, then there is a great chance of generating much more improved and accurate predictions. Normally, these techniques should not be difficult but instead they should be understandable and user friendly. An indicative evaluation of the proposed selection techniques on the user effort and the recommendation accuracy can be seen in Table 3.2.
### Methods

<table>
<thead>
<tr>
<th>Methods</th>
<th>User Effort</th>
<th>Recommendation Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGCN</td>
<td>★★★★★</td>
<td>★★★★☆</td>
</tr>
<tr>
<td>(Log pop)×Ent</td>
<td>★★★</td>
<td>★★★★☆</td>
</tr>
<tr>
<td>Entropy0</td>
<td>★★★★★</td>
<td>★★★★☆</td>
</tr>
<tr>
<td>HELF</td>
<td>★★★</td>
<td>★★★☆</td>
</tr>
<tr>
<td>Popularity</td>
<td>★★★★★</td>
<td>★★★☆</td>
</tr>
<tr>
<td>Item-Item</td>
<td>★★★★★</td>
<td>★★</td>
</tr>
<tr>
<td>Entropy</td>
<td>★</td>
<td>★★</td>
</tr>
<tr>
<td>Random</td>
<td>★</td>
<td>★★</td>
</tr>
</tbody>
</table>

Table 3.2: The evaluation of the proposed selection methods on user effort and on prediction accuracy (5star: Best, 1star: Worst).

For our recommender system we have initially taken into consideration the pure entropy method, which is also referred as non-adaptive method. Non-adaptive methods are able to present similar items to all new users ignoring the existence of changes in knowledge of the user being asked. Pure entropy $H(a_t)$ which is usually characterized by low complexity, represents the scattering of the item ratings in the rating matrix. The basic structure of entropy’s algorithm can be seen in the following figure 3.3.

In this point it is worth mentioning that this method is capable of providing a lot of information for each rating. However, this kind of information is not always really useful as the system can present some items that are totally unknown to the majority of the users.
Function Entropy \((a_i)\)
{
  \text{entropy}(a_i) = 0
  \text{for each item } a_i \text{ in dataset}
  \{
    \text{for } i \text{ as each of the possible rating values } //\text{in case of MovieLens, } i = 1\ldots5
    \{
      \text{if (rating}(a_i) == i):
        \{
          \text{value}[i] += 1 //rating frequencies
        \}
    \}
    \text{proportion}[i] = \text{value}[i]/(\text{total number of users who rate } a_i)
    \text{entropy}(a_i) += \text{proportion}[i]*\text{Math.log(proportion}[i],2)
  \}
  \text{entropy}(a_i) = -\text{entropy}(a_i)
}

Picture 3.3: Algorithm of pure entropy method

The author also examines the Entropy0 method which is the Entropy considering missing values. In the previous method (Pure Entropy), missing ratings were not taken into consideration. In order to address the problem of an item without evaluation, the method of Entropy0 zero is introduced: All missing ratings belong to a new category referred as “0” while “1-5” continues to be the normal rating scale as it was before. The following equation shows the Entropy0 formulation using a weighted approach:

\[
\text{Entropy}0(a_i) = -\frac{1}{\sum_i w_i} \sum_{i=0}^5 p_i w_i \text{log}(p_i) \quad (12)
\]

where \(w_0 = 0.5\) represents the weight for the missing ratings, and \(w_i = 1 \) (for \(i = 1,\ldots5\)) represents the original ratings of the dataset. It is also worth noting that if we change \(w_0\) to 0, the Entropy0 is altered to the Pure Entropy. Entropy0 manages to address some of the drawbacks of Pure Entropy, and make a distinction between the unknown items (items that have a small number of ratings) and frequently rated items. It is also observed that Entropy0 generates better results than Popularity method.
3.2.4 Learning Preferences of New Users in Recommender Systems: An Information Theoretic Approach

This paper [18], is trying to address the cold start problem by examining the effectiveness of several item selection methods based on information theory. The basic concept of this procedure is to use each of these methods in order to find a set of items, and then to evaluate how effective these items are in constructing new users profiles. We also have to note that the author is mainly focused on developing methods based on information theory, aiming to extract information about new users’ habits and tastes. Similar to the previous section, the author also notices that pure entropy has many limitations and as a result he proposes some variations: Entropy0 and HELF. The methods used by the author are: Popularity, Entropy0 (Entropy Considering Missing Values), HELF (Harmonic mean of Entropy and Logarithm of Frequency) and IGCN (Information Gain through Clustered Neighbors).

Entropy and Entropy0 were presented in the previous section. Regarding the other 3 methods we have:

- Popularity shows how frequently the users rate the items, and it is considered a very easy and inexpensive technique.

- HELF which is the Harmonic mean of Entropy and Logarithm of rating Frequency can be formulated with the above equation:

\[
HELF_{a_i} = \frac{2 \times LF'_{a_i} \times H'(a_i)}{LF'_{a_i} + H'(a_i)}
\]  

(13)

where \(LF'_{a_i}\) is the normalized algorithm of the rating frequency of \(a_i\): \(\lg(|a_i|) / \lg(|U|)\) and \(H'(a_i)\) is the normalized entropy of \(a_i\): \(H(a_i) / \lg(5)\).

Finally, ICGN which is the Information Gain through Clustered Neighbors, calculates information gain of items. Additionally, the ratings data is selected according to the users that match best with the active’s user profile until now. The information gain of an item \(a_i\) can be calculated by the below function:

\[
IG(a_i) = H(C) - \sum_{r} \frac{|C'_{a_ir}|}{|C|} H(C'_{a_ir})
\]  

(14)

where \(H(X)\) represents the entropy of a distinct variable \(X\), while \(C\) indicates the distribution of users into clusters defining the number of users that belong to each cluster. \(C'_{a_ir}\) indicates the distribution into classes of those users that have rated the
item \( a_t \) with value \( r \). \( \sum_r \frac{|c_{a_t}^r|}{|C|} H(C_{a_t}^r) \) represents the weighted average of entropies of the partitions of the class distribution (C) caused by the ratings of the item \( a_t \).

The following figure shows the results of the above methods from the offline simulation:

![Graph showing the results of selection methods](image)

**Picture 3.3:** The figure shows how familiar the presented movies are to the users, for each of the aforementioned selection methods.

It is obvious that popularity method selects the most familiar items to users while HELF generates the worst results. Moreover, we can see that Entropy0 is also capable of producing some satisfactory results.

Then, in figure 3.4 we can see the results regarding the accuracy of recommendations. From these results we can see that both IGCN and Entropy0 have a good performance for both of the metrics. However, HELF produces some confusing results because regarding MAE is one of the worst, while regarding Expected Utility is one of the best.
Picture 3.4: The plots show the effectiveness of the generated user’s profiles. (a),(b) present the recommendation accuracy of User-based kNN CF algorithm, while (c),(d) present the accuracy of Item-based kNN CF algorithm. Mean absolute error (MAE) is better for lower values, and Expected Utility is better for higher values.
4 Requirements and Design

In this chapter, we present the functional and non-functional requirements of our implementation, taking into consideration the cold start problem and the overview of the system model that is described briefly below: The system asks the new user for explicit ratings and then, based on the already existing dataset which has the other users’ ratings, generates recommendations for the new user.

Furthermore, we are going to introduce the design of our proposed system and how is it possible to satisfy the aforementioned requirements. We will give a brief description about the architecture of the system, the data that we used, the assumptions we made, and finally the basic execution flow.
4.1 Requirements

4.1.1 Functional requirements

In this section, there will be a brief explanation of the non-functional requirements of the user interaction with the software.

<table>
<thead>
<tr>
<th>FR ID</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Display movies for rating</td>
<td>The system should provide an interface that presents movies to users</td>
</tr>
<tr>
<td>2</td>
<td>Request users demographic data</td>
<td>The system should provide an interface which asks users to give their demographic information</td>
</tr>
<tr>
<td>3</td>
<td>Display movies for rating based on their entropy</td>
<td>The system should be able to calculate movies entropy, and present (for rating) the movies with the highest scores of entropy</td>
</tr>
<tr>
<td>4</td>
<td>Display movies for rating based on users’ demographic data</td>
<td>The system should be able to find the k most common neighbors (based on demographics) for the target user, and present (for rating) their corresponding movies.</td>
</tr>
<tr>
<td>5</td>
<td>Ask user for explicit ratings</td>
<td>The system should provide an interface that asks users for explicit ratings</td>
</tr>
<tr>
<td>6</td>
<td>Generate recommendations using collaborative filtering</td>
<td>The system should be able to generate recommendations by processing the ratings given by the target user and other similar user from the existing dataset.</td>
</tr>
</tbody>
</table>

Table 4.1: Functional requirements of our proposed system
4.1.2 Non-Functional requirements

In this section, there will be a brief explanation of the non-functional requirements of the user interaction with the software

<table>
<thead>
<tr>
<th>NFR ID</th>
<th>Title</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Accurateness</td>
<td>The system should provide precise recommendations which correspond to the target user actual movie preferences</td>
</tr>
<tr>
<td>2</td>
<td>Simple</td>
<td>The system should provide a simple interface that will be understandable and easy to use.</td>
</tr>
<tr>
<td>3</td>
<td>Interesting</td>
<td>The system should not be boring and should be capable of capturing user’s attention during the whole procedure</td>
</tr>
<tr>
<td>4</td>
<td>Fast</td>
<td>The system should be fast, both in terms of producing the recommendations but also in terms of collecting user’s ratings and his demographic information.</td>
</tr>
</tbody>
</table>

Table 4.2: General Non-Functional requirements of our proposed system
The software must also be compliant with ISO 9126 quality characteristics ([http://www.sqa.net/iso9126.html](http://www.sqa.net/iso9126.html)).

<table>
<thead>
<tr>
<th>Category</th>
<th>Subcategory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functionality</td>
<td>Compliance</td>
<td>The system should minimize intrusiveness and be compliant with privacy laws</td>
</tr>
<tr>
<td>Reliability</td>
<td>Maturity</td>
<td>The software should face very rarely failures</td>
</tr>
<tr>
<td></td>
<td>Fault tolerance</td>
<td>Software should be able to withstand and recover from failures</td>
</tr>
<tr>
<td></td>
<td>Recoverability</td>
<td>Ability to bring back a failed system to full operation, including data</td>
</tr>
<tr>
<td>Usability</td>
<td>Understandability</td>
<td>Ease of which the software’s functions can be understood</td>
</tr>
<tr>
<td></td>
<td>Learnability</td>
<td>It should be easy to learn for every kind of user (i.e. no tech savvy)</td>
</tr>
<tr>
<td></td>
<td>Operability</td>
<td>Ability of the software to be easily operated by a given user in a given environment.</td>
</tr>
<tr>
<td>Efficiency</td>
<td>Time Behavior</td>
<td>Response time &lt; 5 sec</td>
</tr>
<tr>
<td></td>
<td>Resource behavior</td>
<td>Use a little amount of memory</td>
</tr>
<tr>
<td>Portability</td>
<td>Adaptability</td>
<td>Characterizes the ability of the system to change to new specifications or operating environments.</td>
</tr>
</tbody>
</table>

Table 4.3: ISO 9126 quality characteristics
4.2 System architecture

The user interacts with the system through an interface that we have developed. The interface is capable of displaying movies to user, and also asking for explicit ratings. It is also asks for users demographic data and store all this information on the disk. Finally the system is capable of generating recommendations based on user’s ratings and his demographic information. This can be implemented by finding the top k most common neighbors and then using collaborative filtering bias subtracted technique.

In order to provide an optimal service to users, the system must be able to generate fast and precise recommendations. Furthermore, the ask-to-rate method should be simple and not boring, but instead should capture user’s attention throughout the duration of the whole procedure of information gathering.

The aforementioned functional and non-functional requirements should be satisfied by designing a recommender system with various components and operations. Below, we provide the key points of its operation:

- **Input:** MovieLens 100k Dataset, new user movies ratings and his demographic data
- **Output:** Generated recommendations based on: 1) select movies for rating randomly, 2) select movies for rating, considering users demographic data, 3) select movies for rating, considering their entropy score, 4) combination of 2, 3 methods.
- **Basic steps of the systems functionality:**
  1. User starts the system
  2. System loads the MovieLens 100k dataset
  3. Target user gives his demographic data – optional (only in case the algorithm requests user’s demographic data)
  4. System calculates movies entropy – optional (only in case the algorithm take into consideration movies entropy)
  5. System displays movies for rating
  6. User rates movies
  7. System generates and displays recommendations
  8. User exits the system
In this point it is worth mentioning that our proposed system consists of four independent and different algorithms (scripts):

1. The first algorithm (basic algorithm) displays movies for rating randomly, and then generates recommendations based on collaborative filtering.
2. The second algorithm (demographic based algorithm) displays movies for rating based on users demographic data, and then generates recommendations like the basic script.
3. The third algorithm (entropy0 based algorithm) displays movies for rating based on their entropy0 scores (movies with the highest entropy0 scores are presented first), and then generates recommendations like the basic script.
4. The fourth algorithm (demographic and entropy0 based algorithm) which is a combination of the third and the fourth script, displays movies for rating based on users demographics and movies entropy0 scores, and then generates recommendations like the basic script.

4.2.1 Recommendation engine

As we have mentioned above, the recommendation engine is implemented by using the kNN algorithm and bias subtracted user based collaborative filtering, and the users similarities are calculated by the Pearson correlation scheme. In other words, we will have the user-similarity matrix that includes the similarities (Pearson correlation) between users. Also the recommendations are calculated by taking into account only the top-k most similar users (kNN algorithm). Finally, in the collaborative filtering technique we have managed to prevent biases related with the users by subtracting each user’s average rating from each user’s rating, and then add that average at the end (bias subtracted collaborative filtering).

4.3 Used dataset

For this project we have used the MovieLens 100k dataset as we have already mentioned in previous chapters. This dataset, was developed by GroupLens Research Project and contains 1682 movies with 100,000 ratings on the rating scale 1-5 provided by 943 users. Moreover, it is based on explicit information given by users during their sign-up on MovieLens website. It is also worth mentioning that every user included in the dataset has rated at least 20 movies.
The MovieLens dataset contains a lot of files, however for our implementation we have singled out and used the following files:

- **u.user**: This file contains demographic data about users. This information has the following format:

  user id | age | gender | occupation | zipcode.

  From the above structure, the zipcode feature was removed because it is not needed for our project. Below we present the first 3 users of the u.user file:

<table>
<thead>
<tr>
<th>user id</th>
<th>age</th>
<th>gender</th>
<th>occupation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>24</td>
<td>M</td>
<td>Technician</td>
</tr>
<tr>
<td>2</td>
<td>53</td>
<td>F</td>
<td>Other</td>
</tr>
<tr>
<td>3</td>
<td>23</td>
<td>M</td>
<td>Writer</td>
</tr>
</tbody>
</table>

  Table 4.4: A sample containing the first 3 rows of the u.user file

The users are classified based on their demographic data by converting the above features (age, gender and occupation) to numeric scales in order to estimate their similarity. As a result we have:

1. **a)** Age is represented within the ranges: 0-18, 19-24, 25-30, 31-40, 41-50, 51-60, 61-70, 71-100. So for a 24 year old user, we have value 1 for the 19-24 age range and 0s for the other ranges.
2. **b)** Gender is specified by 0 and 1
3. **c)** Occupation is also specified by 0s and 1

Combining the above features we develop the below model:

- ✓ **age** = ['18', '24', '30', '40', '50', '61', '70', '100']
- ✓ **gender** = ['M', 'F']
- ✓ **occupation** = ['administrator', 'artist', 'doctor', 'educator', 'engineer', 'entertainer', 'executive', 'healthcare', 'homemaker', 'lawyer', 'librarian', 'marketing', 'none', 'other', 'programmer', 'retired', 'salesman', 'scientist', 'student', 'technician', 'writer']
- ✓ **combined_features** = ['18|0', '24|1', '30|2', '40|3', '50|4', '60|5', '70|6', '100|7', 'm|8', 'f|9', 'administrator|10', 'artist|11', 'doctor|12', 'educator|13', 'engineer|14', 'entertainer|15', 'executive|16', 'healthcare|17', 'homemaker|18', '...']
'lawyer|19', 'librarian|20', 'marketing|21', 'none|22', 'other|23', 'programmer|24', 'retired|25', 'salesman|26', 'scientist|27', 'student|28', 'technician|29', 'writer|30' [71x46] [143x760]

Applying the same logic as the author of 3.2.2 in chapter 3, the above data is used for the calculation of demographic correlations by taking into consideration the user vector similarities. The user demographic vector is defined as a vector with 31 features and can be seen in detail in the following table:

<table>
<thead>
<tr>
<th>feature #</th>
<th>feature contents</th>
<th>comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>age &lt;= 18</td>
<td>• each user belongs to a single age group,</td>
</tr>
<tr>
<td>1</td>
<td>18 &lt; age &lt;= 24</td>
<td>• the corresponding slot takes value 1 (true)</td>
</tr>
<tr>
<td>2</td>
<td>24 &lt; age &lt;= 30</td>
<td>• the rest of the features remain 0 (false)</td>
</tr>
<tr>
<td>3</td>
<td>30 &lt; age &lt;= 40</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>40 &lt; age &lt;= 50</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>50 &lt; age &lt;= 60</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>60 &lt; age &lt;= 70</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>70 &lt; age &lt;= 100</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Male</td>
<td>• the slot describing the user gender is 1</td>
</tr>
<tr>
<td>9</td>
<td>Female</td>
<td>• the other slot takes a value of 0</td>
</tr>
<tr>
<td>10-30</td>
<td>occupation</td>
<td>• a single slot describing the user occupation is 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• the rest of the slots remain 0</td>
</tr>
</tbody>
</table>

Table 4.5: Description of the user demographic vector

For example, considering the first user of the table 4.4 (24 year old, Male technician) we can see that the corresponding combined_features list contains the below values:

[0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
• **u.data**: This is our main dataset that includes 100000 ratings for 1682 movies given by 943 users. Information is formatted on a table with four columns (user id, movie id, rating, timestamp). Users are numbered consecutively from 1 to 943 and every user id is unique for each user. Moreover the same logic applies for movies (movie id ranges from 1 to 1682). Also as we have mentioned before, ratings range from 1 to 5. Finally the timestamps are unix seconds since 1/1/1970 UTC, but in our implementation we will not take them into consideration. Below we give a small sample of the first 5 rows of this dataset where we have omitted the timestamp column as we mentioned before:

<table>
<thead>
<tr>
<th>user id</th>
<th>movie id</th>
<th>rating value</th>
</tr>
</thead>
<tbody>
<tr>
<td>196</td>
<td>242</td>
<td>3</td>
</tr>
<tr>
<td>186</td>
<td>302</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>377</td>
<td>1</td>
</tr>
<tr>
<td>244</td>
<td>51</td>
<td>2</td>
</tr>
<tr>
<td>166</td>
<td>346</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.6: A sample containing the first 5 rows of the u.data file

• **u.item**: Finally, this file includes the metadata for our movies. More specifically it is formatted as a table with rows where each row corresponds to a movie. Also the table has 23 columns where: the 1\textsuperscript{st} column is the movie id, the 2\textsuperscript{nd} is the movie title, the 3\textsuperscript{rd} is the release date of the movie, the 4\textsuperscript{th} is the imdb link for this movie, and the remaining columns corresponds to the movie genres. All the possible genres can be: | unknown | Action | Adventure | Animation | Children's | Comedy | Crime | Documentary | Drama | Fantasy | Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi | Thriller | War | Western |. A 1 indicates the movie is of that genre, while a 0 indicates it is not. Also movies can be in several genres at once. It is also worth noting that the movie ids are the ones that are used in the u.data dataset. For our implementation we care only about the titles, as a result we keep only the first two columns (movie id and movie title). Below we give a small sample of the first 5 rows of this dataset:
<table>
<thead>
<tr>
<th>movie id</th>
<th>movie title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Toy Story (1995)</td>
</tr>
<tr>
<td>2</td>
<td>GoldenEye (1995)</td>
</tr>
<tr>
<td>3</td>
<td>Four Rooms (1995)</td>
</tr>
<tr>
<td>4</td>
<td>Get Shorty (1995)</td>
</tr>
<tr>
<td>5</td>
<td>Copycat (1995)</td>
</tr>
</tbody>
</table>

Table 4.7: A sample containing the first 5 rows of the u.item file
4.4 Assumptions

For the sake of simplicity, and in order to overcome various problems, we are going to make some assumptions that are applied within the constraints of our project. The assumptions are described as follows:

- The u.item dataset which contains metadata about movies, remains stable and does not change from the start till the end of our implementation. In other words, there are not new movies that are added, or there are not movies characteristics that are altered.
- The u.users dataset that contains user’s demographic information does not change. This means that neither new users are added nor existing users demographic data changes throughout the whole procedure of our implementation. Even if a new user is introduced to the system, his demographic data is saved only temporarily.
- The same logic is also applied for the u.data dataset, as all the including information (users’ movies ratings) does not change. Similarly, in case a new user enters the system, his movies ratings are only saved temporarily and are discarded when the user exits the system.
- There is always a fixed number of movies displayed to target user for rating.
- The number of recommended movies is also fixed.
- Each algorithm (script) provides recommendations when user has rated 10 movies.

4.5 Flowchart diagrams

The following sections provide a brief description of the functions of the four algorithms that were discussed in 4.2: Basic algorithm, Demographics based algorithm, Entropy0 based algorithm and Demographics-Entropy0 based algorithm. Moreover we introduce the procedure of inserting user’s demographics and providing ratings for movies.
4.5.1 **Basic algorithm flowchart diagram**

In this section we introduce the flowchart of the basic algorithm which only uses collaborative filtering. Movies are displayed for rating randomly without using a specific approach.

![Flowchart Diagram](image)

Picture 4.1 Basic algorithm
4.5.2 User Demographics based algorithm flowchart diagram

In this section we introduce the flowchart of the user demographics based algorithm which displays movies for rating based on users’ demographic data and uses collaborative filtering for recommendations.

Picture 4.2: User Demographics based algorithm
Below we present a screenshot of how the system asks user to insert his demographic data:

![Screenshot of the system asking for age, gender, and occupation](image)

Choose the corresponding number: 3
Select your gender:
1. Male
2. Female

Choose the corresponding number: 15
Select your occupation:
1. administrator
2. artist
3. doctor
4. educator
5. engineer
6. entertainer
7. executive
8. healthcare
9. homemaker
10. lawyer
11. librarian
12. marketing
13. none
14. other
15. programmer
16. retired
17. salesman
18. scientist
19. student
20. technician
21. writer

Choose the corresponding number: 15

Picture 4.3: System asks target user to insert his age, gender and occupation
4.5.3 Movies Entropy0 based algorithm flowchart diagram

In this section we introduce the flowchart of the movies entropy0 based algorithm which displays movies for rating based on their entropy0 scores and uses collaborative filtering for recommendations.

![Flowchart Diagram]

Picture 4.4: Movies Entropy0 based algorithm
4.5.4 User Demographics and movies entropy0 based algorithm flowchart diagram

In this section we introduce the flowchart of the user demographics and movies entropy0 based algorithm which displays movies for rating based on users’ demographics and movies entropy0 scores and uses collaborative filtering for recommendations.

![Flowchart Diagram](image)

Picture 4.5: Users demographics and Movies Entropy0 based algorithm
4.5.5 Movies rating function flowchart diagram

In this point we provide the flowchart diagram for the movie rating procedure. User starts the system and the system displays movies for rating. User either choses a movie for rating or presses -1 to load a new movielist. If he has rated 10 movies, the system provides recommendations by using collaborative filtering. On the other hand, if he has rated less than 10 movies the user either changes the movielist or keeps the already existing movielist.

![Flowchart Diagram]

Picture 4.6: Movies rating function
Below we present an example of how movies are displayed by the system to the target user for rating:

```plaintext
[1: 'Toy Story (1995)'
[3: 'Four Rooms (1995)'
[4: 'Get Shorty (1995)'
[5: 'Copycat (1995)'
[6: 'Shanghai Triad (Yao a yao yao dao wai po qiao) (1995)'
[7: 'Twelve Monkeys (1995)'
[8: 'Babe (1995)'
[9: 'Dead Man Walking (1995)'
[11: 'Seven (Se7en) (1995)'
[14: 'Il Postino, Il (1994)'
[15: 'Mr. Holland's Opus (1995)'
[16: 'French Twist (Gazon maudit) (1995)'
[17: 'From Dusk Till Dawn (1996)'
[20: 'Angels and Insects (1995)'
```

Choose a movie, or press -1 to change movieset:

Picture 4.7: Movies displayed to user for rating
4.5.6 Inserting target user demographics

In the following flowchart we describe how user inserts his demographic data. First of all he inserts his age, then he inserts his gender and finally he enters his occupation. Finally the system uses the kNN algorithm to find user’s top k most common neighbors based on demographics.

![Flowchart](image)

Picture 4.8: Insert users demographics function
5 Implementation

5.1 Basic parts of our implementation

As we have already mentioned in the previous chapter, we developed four different scripts: basic, demographic-based, entropy0-based and demographic & entropy0-based. The aforementioned scripts were developed in Python (version 3.6) language using the PyCharm development environment. Furthermore in order to manage some important Python packages we have used the Anaconda package manager. Below we are going to give a brief description of the main functions of the four scripts:

5.1.1 Functions of the Basic script

- `readFullDataset(dataSetFilePath)`: This function reads the full dataset (u.data) which contains the 100000 ratings. It takes one argument (dataSetFilePath) of type string, which is the path of the u.data file and finally returns the full dataset as Pandas dataframe with the following names as columns: 'user_id', 'item_id', 'rating', 'timestamp'

- `readMovieSet(movieSetFilePath)`: This function reads the movie dataset (u.item) which contains information about movies. It takes one argument (movieSetFilePath) of type string, which is the path of the u.item file and finally returns the movie dataset as Pandas dataframe with the following names as columns: 'item_id', 'title'

- `insertNewUserRatings(ids_titles, fullDataSet, newUserID, timestamp, known_positives, mySelMovies)`: This function displays movies to the new user and asks for explicit ratings. It takes 6 arguments: `ids_titles` from the readMovieSet(movieSetFilePath) function, `fullDataSet` from the readFullDataset(dataSetFilePath) function, the `newUserID`, a random `timestamp`, and `known_positives, mySelMovies` are both Python lists. Finally returns the full dataset that includes the new user with his ratings.
• `numberOfUsers(fullDataSet)`: This function returns the number of users in the dataset. It takes the full dataset as an argument.

• `numberOfMovies(fullDataSet)`: This function returns the number of movies in the dataset. It takes the full dataset as an argument.

• `getUserItemMatrix(n_users, n_items, fullDataSet)`: This function returns the user-item matrix. It takes the number of users, the number of movies and also the full dataset as arguments.

• `calculateUsersPearsonCorrelation(user_item_matrixTrain)`: This function returns a matrix with the users Pearson correlation based on their ratings. It takes the user-item matrix as an argument.

• `predict_Top_K_no_Bias(ratings, similarity, k=40)`: This function returns a numpy array with the predictions for each user (kNN Collaborative filtering-bias subtracted). It takes the user-item matrix, the matrix with Users Pearson Correlation and a number for the kNN algorithm (default is 40) as arguments.

• `printPredictedMoviesUserBased(user, n)`: This function prints the top–n recommended movies for a given user id. It takes the user id and the number of recommended movies as arguments.

### 5.1.2 Functions of the Demographic-based script

• `_read_raw_data(path)`: This function reads the demographic data of the existing users. It takes one argument (path) of type string, which is the path of the zip file that contains all the datasets and finally returns the demographic data.

• `createUserMetaDataList (users_raw, users_age, users_occup, user_meta_raw)`: This function asks the new user for his demographic data (age, gender and occupation) and then append this data to the previous dataset with the users demographics. It takes 4 lists as arguments: the first 3 lists are from the demographic model that we have created, and the 4th list is demographics of the dataset.
- parse_user_metadata (num_users, user_meta_raw, users_combined_features): This function transforms users demographics list (with the new user) into a list with zeros and ones and return this list. It takes as arguments the number of users, the demographics of all users (including the new user), and the user’s combined features from the demographic model.

- euclideanDistance(instance1, instance2, length): This function calculates and returns the Euclidean distance of two instances (instance 1 and instance 2).

- getNeighbors(trainingSet, testInstance, k): This function calculates and returns the k most common neighbors of a specified testInstance from a given dataset.

- readFullDataset(dataSetFilePath): The same as Basic script.

- readMovieSet(movieSetFilePath): The same as Basic script.

- insertNewUserRatings(ids_titles, fullDataSet, newUserID, timestamp, known_positives, mySelMovies, neighborsmovies): The same as Basic script, taking into consideration the most common neighbors movies.

- numberOfUsers(fullDataSet): The same as Basic script

- numberOfMovies(fullDataSet): The same as Basic script

- getUserItemMatrixDemographicsBased(n_users, n_items, fullDataSet, neighbors) The same as the getUserItemMatrix of the Basic script, taking into consideration only the common neighbors.

- getUserItemMatrix(n_users, n_items, fullDataSet): The same as Basic script

- calculateUsersPearsonCorrelation(user_item_matrixTrain): The same as Basic script

- predict_Top_K_no_Bias(ratings, similarity, k=40): The same as Basic script

- printPredictedMoviesUserBased(user, n): The same as Basic script
5.1.3 Functions of the Entropy0-based script

- `readFullDataset(dataSetFilePath)`: The same as Basic script.

- `readMovieSet(movieSetFilePath)`: The same as Basic script.

- `insertNewUserRatings(ids_titles, fullDataSet, newUserID, timestamp, known_positives, mySelMovies, entropy_indexes)`: The same as Basic script taking into consideration movies entropy (entropy_indexes).

- `numberOfUsers(fullDataSet)`: The same as Basic script

- `numberOfMovies(fullDataSet)`: The same as Basic script

- `calcMoviesEntropy0(fullDataSet, n_users, n_items, neighbors)`: This function calculates and returns the entropy0 values of the full dataset. It takes as arguments the full dataset and the number of users and movies.

- `getUserItemMatrix(n_users, n_items, fullDataSet)`: The same as Basic script

- `calculateUsersPearsonCorrelation(user_item_matrixTrain)`: The same as Basic script

- `predict_Top_K_no_Bias(ratings, similarity, k=40)`: The same as Basic script

- `printPredictedMoviesUserBased(user, n)`: The same as Basic script
5.1.4 Functions of the Demographic & Entropy0-based script

- \(_{read\_raw\_data}(path)\): The same as Demographic based script.

- createUserMetaDataList \((users\_raw, users\_age, users\_occup, user\_meta\_raw)\): The same as Demographic based script.

- \(_{parse\_user\_metadata}(num\_users, user\_meta\_raw, users\_combined\_features)\): The same as Demographic based script.

- \(euclideanDistance(instance1, instance2, length)\): This function calculates and returns the Euclidean distance of two instances (instance 1 and instance 2).

- getNeighbors(trainingSet, testInstance, k): This function calculates and returns the k most common neighbors of a specified testInstance from a given dataset.

- \(readFullDataset(dataSetFilePath)\): The same as Basic script.

- \(readMovieSet(movieSetFilePath)\): The same as Basic script.

- insertNewUserRatings(ids_titles, fullDataSet, newUserID, timestamp, known_positives, mySelMovies, entropy_indexes): The same as Basic script.

- numberOfUsers(fullDataSet): The same as Basic script

- numberOfMovies(fullDataSet): The same as Basic script

- calcMoviesEntropy0(fullDataSet, n_users, n_items, neighbors): The same as Entropy0 based script.

- getUserItemMatrix(n_users, n_items, fullDataSet): The same as Basic script

- calculateUsersPearsonCorrelation(user_item_matrixTrain): The same as Basic script

- predict_Top_K_no_Bias(ratings, similarity, k=40): The same as Basic script

- printPredictedMoviesUserBased(user, n): The same as Basic script
5.2 Basic flow of each script

Below we are going to present and highlight the most important points of the basic flow of each script. The corresponding scripts can be seen in detail in the Appendix (source code section).

5.2.1 Basic script

- Lines 198-201: The system initializes the new user and also some lists.
- Lines 203-206: The system reads the full dataset and the full movieset.
- Lines 209-213: The system displays movies and asks for explicit ratings, then it calculates the number of users and movies.
- Line 216: The system creates the user item matrix taking into consideration the new user.
- Line 219: The system calculates users similarity by Pearson correlation.
- Line 222: The system generates predictions for users.
- Line 238: The system prints the top 10 recommended movies for the new user.

5.2.2 Demographic-based script

- Lines 369-372: The system initializes the new user and also some lists.
- Line 375: The system fetch the users demographic data.
- Lines 378-392: The system creates the model for the demographic based system by creating some lists.
- Line 395: The system asks for the demographic data of new user and appends it to the list with the demographics of existing users.
- Lines 397-400: The system reads the full dataset and the full movieset.
- Line 404: The system creates a list with zeros and ones that corresponds to users demographic data including the new user.
- Line 406: The system creates a list with zeros and ones that corresponds to users demographic data without the new user.
- Line 408: The system creates a list with zeros and ones that corresponds only to new user demographic data.
- Line 411: The system finds the 20 most common neighbors of the new user, taking into consideration users demographics.
- Lines 413-415: The system calculates the number of users and movies.
5.2.3 Entropy0-based script
- Lines 253-256: The system initializes the new user and also some lists.
- Lines 258-261: The system reads the full dataset and the full movieset.
- Line 263-265: The system calculates the number of users and movies
- Line 267: The system calculate movies ratings entropy0 values and return movies indexes starting from the highest entropy0 values to the lowest
- Lines 271: The system displays movies and asks for explicit ratings, considering movies entropy0 values.
- Line 273-275: The system re-calculates the number of users and movies.
- Line 278: The system creates the user item matrix taking into consideration the new user.
- Line 281: The system calculates users similarity by Pearson correlation
- Line 284: The system generates predictions for users
- Line 300: The system prints the top 10 recommended movies for the new user

5.2.4 Demographic & Entropy0-based script
- Lines 408-411: The system initializes the new user and also some lists.
- Line 414: The system fetch the users demographic data
- Lines 417-431: The system creates the model for the demographic based system by creating some lists
- Line 433: The system asks for the demographic data of new user and appends it to the list with the demographics of existing users.
- Line 437: The system creates a list with zeros and ones that corresponds to users demographic data including the new user
- Line 439: The system creates a list with zeros and ones that corresponds to users demographic data without the new user
- Line 441: The system creates a list with zeros and ones that corresponds only to new user demographic data.
- Line 444: The system finds the 20 most common neighbors of the new user, taking into consideration users demographics.
- Lines 446-449: The system reads the full dataset and the full movieset.
- Lines 451-453: The system calculates the number of users and movies
- Line 457: The system calculates movies ratings entropy0 values taking into consideration the ratings of the 20 most common neighbors, and returns the corresponding movies indexes from the highest to the lowest.
- Lines 460: The system displays movies and asks for explicit ratings, taking into consideration the movies indexes of the previous line
- Line 462-464: The system calculates the number of users and movies
- Line 467: The system creates the user item matrix taking into consideration the new user.
- Line 470: The system calculates users similarity by Pearson correlation
- Line 473: The system generates predictions for users
- Line 489: The system prints the top 10 recommended movies for the new user
6 Evaluation and future work

6.1 Results and evaluation

In this part, we proceed in the evaluation of our system: We have tested each of the four scripts on 25 different users. 10 of them were female and 15 were male with their age ranging from 20-60.

In order to remove bias from the evaluation, we do not disclose in which of the four scripts each predicted movie set corresponds to. For that reason, we modified the `printPredictedMoviesUserBased()`, function of each script by commenting out the last line and adding another line that saves the predicted movie set in excel format. As a result, the modified function will be:

```python
def printPredictedMoviesUserBased(user, n):
    user = user - 1
    n = n - 1
    pred_indexes = [i + 1 for i in np.argsort(-user_prediction_user[user])]
    pred_indexes = [item for item in pred_indexes if item not in known_positives]
    movies_ids_titles = pd.read_csv('u.item', sep='|', header=None, encoding='latin-1', names=['itemID', 'title'], usecols=[0, 1])
    pd_pred_indexes = pd.DataFrame(pred_indexes, columns=['itemID'])
    pred_movies = pd.merge(pd_pred_indexes, movies_ids_titles, on='itemID')
    print('Modified version of the printPredictedMoviesUserBased() function
As we can see from the above figure, the blue arrow indicates the added line of code that writes the predicted movie set into an excel file, while the red arrow indicates the commented code. By this way, the program hides the predicted movie set and the user is completely unaware of which predicted movie set corresponds to each of the 4 scripts.

The evaluation process is presented as follows: First of all, every user have to execute each of the four scripts (systems). Then, the four predicted movie sets are shown to the user: The A set corresponds to the demographic-based script, the B set corresponds to the entropy0-based script, the C set corresponds to the basic script and finally the D set corresponds to the demographic & entropy0 based script, however the user is not aware of that matching in order to remove bias as we have mentioned before. Finally, the users have to rank their most preferred to least preferred movie set by declaring his preferences.
```
For example: Firstly I choose movieset B, secondly I choose movieset A, then I choose movieset D and finally I choose movieset C. The first preference is awarded with four points, the second with 3, the third with 2 and the final with 1 point. By this way we can get the required preference scores for each of the 4 scripts in order to complete the evaluation.

In order to calculate the final scores of the evaluation for each script, we created an excel file with all the required information. This can be seen below:

<table>
<thead>
<tr>
<th>User info</th>
<th>User preference scores for each system</th>
<th>User effort time (min.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>Gender</td>
<td>Occupation</td>
</tr>
<tr>
<td>27</td>
<td>F</td>
<td>Student</td>
</tr>
<tr>
<td>55</td>
<td>F</td>
<td>Technician</td>
</tr>
<tr>
<td>54</td>
<td>M</td>
<td>Technician</td>
</tr>
<tr>
<td>30</td>
<td>M</td>
<td>Scientist</td>
</tr>
<tr>
<td>20</td>
<td>F</td>
<td>Artist</td>
</tr>
<tr>
<td>22</td>
<td>M</td>
<td>Student</td>
</tr>
<tr>
<td>25</td>
<td>F</td>
<td>Programmer</td>
</tr>
<tr>
<td>30</td>
<td>M</td>
<td>Engineer</td>
</tr>
<tr>
<td>31</td>
<td>M</td>
<td>Doctor</td>
</tr>
<tr>
<td>18</td>
<td>M</td>
<td>Student</td>
</tr>
<tr>
<td>62</td>
<td>F</td>
<td>Retired</td>
</tr>
<tr>
<td>65</td>
<td>M</td>
<td>Retired</td>
</tr>
<tr>
<td>38</td>
<td>M</td>
<td>Programmer</td>
</tr>
<tr>
<td>31</td>
<td>F</td>
<td>Lawyer</td>
</tr>
<tr>
<td>27</td>
<td>M</td>
<td>Salesman</td>
</tr>
<tr>
<td>26</td>
<td>M</td>
<td>Healthcare</td>
</tr>
<tr>
<td>28</td>
<td>F</td>
<td>Educator</td>
</tr>
<tr>
<td>27</td>
<td>M</td>
<td>Marketing</td>
</tr>
<tr>
<td>25</td>
<td>M</td>
<td>Salesman</td>
</tr>
<tr>
<td>24</td>
<td>F</td>
<td>Educator</td>
</tr>
<tr>
<td>29</td>
<td>M</td>
<td>Engineer</td>
</tr>
<tr>
<td>33</td>
<td>M</td>
<td>Programmer</td>
</tr>
<tr>
<td>35</td>
<td>F</td>
<td>Homemaker</td>
</tr>
<tr>
<td>25</td>
<td>M</td>
<td>Technician</td>
</tr>
<tr>
<td>21</td>
<td>F</td>
<td>Artist</td>
</tr>
</tbody>
</table>

| SUM | 72 | 77 | 50 | 51 | 116 | 104 | 196 | 355 |
| AVG | 3,13 | 3,34 | 2,17 | 2,21 | 5,04 | 4,52 | 8,52 | 15,43 |

Table 6.1: Evaluation table containing user demographics, the preference scores for every script and also the effort time to complete each script
As we can see, demographic-based and entropy0 based systems are topping the list of most preferred systems. Entropy0 based is in the first position with average preference score 3,34, while demographic based comes second with 3,13. Surprisingly, the demographic and entropy0 based system is in the third position with 2,21 average preference score, while the basic system (random selection) is the last with 2,17.

In terms of user effort time, entropy0 based system is again the first with 4,52 minutes average user effort time, demographic based is second with 5,04 minutes, basic system is third with 8,52 minutes while demographics and entropy0 based system is the last with 15,43 minutes (double than that of basic).

Finally, excluding the system that combines demographics and entropy0, we compare and contrast entropy0 based and demographic based systems with the basic system. As a result we have the following table:

<table>
<thead>
<tr>
<th></th>
<th>Demographics based</th>
<th>Entropy0 based</th>
<th>Basic</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM</td>
<td>72</td>
<td>77</td>
<td>50</td>
</tr>
<tr>
<td>Percentage</td>
<td>36,18%</td>
<td>38,69%</td>
<td>25,12%</td>
</tr>
</tbody>
</table>

Table 6.2: Comparing the best two systems with the basic in terms of user preference score

### 6.2 Conclusions

Taking into consideration the above evaluations, we conclude that ratings entropy0 values as well as users demographics can play an important role in addressing cold start problem on collaborative filtering systems. Users tend to prefer entropy0 based systems, while demographics based systems rank second with small difference from the first. Surprisingly, the system that combines both demographics and entropy0 is in the third place with almost the same user preference score as the basic system that offers random selection.

More specifically, the entropy0 based system not only is first in terms of user preference scores, but also requires less user effort than the other three systems. The average user effort for entropy0 based system is 4,5 minutes while demographics based system requires 5 minutes on average. Users seemed happy as both of the aforementioned systems and especially entropy0 based displayed movies that were known to them. Moreover, they were a little concerned about the demographic based system because it displayed for rating only a small portion of the total available movies of the dataset. Furthermore, the system that combines both demographics and entropy0 values, is in the last position.
because it mostly displays unknown movies, so the users have to make a great effort in order to complete this test.

To sum up, we can see that entropy0 based system is in the first place both in terms of users preference scores and also in terms of effort time. This can also be depicted better if we see the tables 6.1 and 6.2 where entropy0 based system is better than the basic by 13% while it requires much less user effort (the average user effort for entropy0 based is 4.5 minutes while for the basic is 8.5 minutes). By this way, we have significantly improved the collaborative movie recommender system both in terms of user preference scores and also in user effort time.

6.3 Future work

As we have seen from the evaluation part, the majority of users were complaining that many movies were old and completely unknown to them. As a result, in a future work there a need to use newer dataset with more known movies. Furthermore, except from the release year, the movie-set should be enhanced with more demographic features such as income level, marital status, number of children, religion etc. By this way we can better understand how much demographics affect recommender systems.

Another idea for future research is to use completely different datasets that include other kind of products such as music, books, electronic devices and more (this can be achieved by using Amazon datasets).

Finally, another addition that will surely attract a lot of attention is to develop recommender systems for mobile devices. As we can see, in the past 10 years there is a rapid growth of mobile devices and more and more people are using smartphones and tablets. As a result, there is a need to develop recommender systems that fit in these mobile devices. The main idea is that the user will send all the required data from his mobile device using an easy and friendly graphical user interface. Then the recommendation engine running on a powerful server will receive and process the user input, generate recommendation and send it back to user’s device. By this way the user will be able to see the predictions through a simple and friendly interface of his mobile device.
Bibliography


8. G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems a survey of the state-of-the-art and possible extensions


23. Oliver Sutton, (2012). Introduction to k Nearest Neighbor Classification and Condensed Nearest Neighbor Data Reduction

Appendix

Source Code

```python
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import pairwise_distances
from random import randint

# We read in the u.data file, which contains the full dataset.
def readFullDataset(dataSetFilePath):
    header = ['user_id', 'item_id', 'rating', 'timestamp']
    return pd.read_csv(dataSetFilePath, sep='\t', names=header)

# We read the the movies titles from the movie dataset
def readMovieSet(movieSetFilePath):
    df_ids_titles = pd.read_csv(movieSetFilePath, sep='|', header=None, encoding='latin-1', names=['itemId', 'title'], usecols=[0, 1])
    ids_titles = np.empty(1682, dtype=np.object)
    for line in df_ids_titles.itertuples():
        ids_titles[line[0]] = line[2]
    return ids_titles

# Insert new user by creating gui in python console
def insertNewUserRatings(ids_titles, fullDataSet, newUserID, timestamp, known_positives, mySelMovies):
    i=0
    j=20
    f=0
    while(f < 10):
        userList = []
        for x in range(i, j):
            userList.append({x%20+1: ids_titles[randint(0, 1681)]})
```
```python
print(\n')
for p in userList:
    print(p)
print(\n')
while (True):
    try:
        var = int(input("Choose a movie, or press -1 to change movieset: "))
    except ValueError:
        print("Wrong input, please insert an integer")
    if ((var<-1 or var>20) and var ==0):
        print("Value must be -1 OR between 1 and 20. Please insert a valid integer")
    else:
        if (1<=var and 20>=var):
            selMovie = str(ids_titles.tolist().index(userList[var-1][var]) + 1)
            if selMovie in mySelMovies:
                print("You have already selected that movie, please choose another movie")
                continue
            mySelMovies.append(str(ids_titles.tolist().index(userList[var-1][var]) + 1))
            break
        if (var == -1):
            if ((1681 - j) >= 20):
                i = j
                j += 20
            elif ((1681 - j) > 0):
                i = j
                j = 1682
            else:
                i = 0
                j = 20
            continue
        else:
            print(\n')
            print("You selected the movie: " + userList[var-1][var] + " with ID: " + str(ids_titles.tolist().index(userList[var-1][var]) + 1))
            print(\n')
            while (True):
                try:
                    rating = int(input("Rate the movie: "))
                except ValueError:
                    print("Wrong input, please insert an integer")
                if (rating < 1 or rating > 5):
                    print("Value must be between 1 and 5. Please insert a valid integer")
                    continue
            break
        known_positives.append(ids_titles.tolist().index(userList[var-1][var]) + 1)
        fullDataSet.loc[len(fullDataSet)] = [newUserID, ids_titles.tolist().index(userList[var-1][var]) + 1, rating, timestamp]
        f = f + 1
        while(f < 10):
        ```
while (True):
    try:
        ch = int(input("To change the movieset press -1, to keep press 1: "))
    except ValueError:
        print("Wrong input, please insert an integer")
        continue
    if (ch != 1 and ch != -1):
        print("Value must be 1 or -1. Please insert a valid integer")
        continue
    break
    if(int(ch) == -1):
        break
    else:
        print('
')
        for p in userList:
            print(p)
        print('
')
        while (True):
            try:
                var = int(input("Choose a movie: "))
            except ValueError:
                print("Wrong input, please insert an integer")
                continue
            if ((var < -1 or var > 20)): 
                print("Value must be between 1 and 20. Please insert a valid integer")
                continue
            selMovie = str(ids_titles.tolist().index(userList[var - 1][var]) + 1)
            if selMovie in mySelMovies:
                print("You have already selected that movie, please choose another movie")
                continue
            mySelMovies.append(str(ids_titles.tolist().index(userList[var - 1][var]) + 1))
            break
            print('
')
            print("You selected the movie: " + userList[var - 1][var] + " with ID: " + str(ids_titles.tolist().index(userList[var - 1][var]) + 1))
            print('
')
            while (True):
                try:
                    rating = int(input("Rate the movie:
"))
                except ValueError:
                    print("Wrong input, please insert an integer")
                    continue
                if (rating < 1 or rating > 5):
                    print("Value must be between 1 and 5. Please insert a valid integer")
                    continue
                break
            known_positives.append(ids_titles.tolist().index(userList[var - 1][var]) + 1)
fullDataSet.loc[len(fullDataSet)] = [newUserID, ids_titles.tolist().index(userList[var - 1][var]) + 1, rating, timestamp]

    f = f + 1
    if((1681-j) >= 20):
        i = j
        j += 20
    elif((1681-j) > 0):
        i = j
        j = 1682
    else:
        i = 0
        j = 20
    print('\n')
return fullDataSet

#------------------------------------------#

#we count the number of unique users and movies.

def numberOfUsers(fullDataSet):
    n_users = fullDataSet.user_id.unique().shape[0]
    return n_users

def numberOfMovies(fullDataSet):
    n_items = fullDataSet.item_id.unique().shape[0]
    return n_items

#------------------------------------------#

#we create user-item matrix

def getUserItemMatrix(n_users, n_items, fullDataSet):
    user_item_matrix = np.zeros((n_users, n_items))
    for line in fullDataSet.itertuples():
    return user_item_matrix

#------------------------------------------#

#we use the pairwise_distances function from sklearn to calculate the pearson correlation

def calculateUsersPearsonCorrelation(user_item_matrixTrain):
    user_similarityPearson = 1 - pairwise_distances(user_item_matrixTrain, metric='correlation') #943*943
    user_similarityPearson[np.isnan(user_similarityPearson)] = 0
    return user_similarityPearson

#------------------------------------------#
#make predictions combining Top-K neighbors and Bias-subtracted collaborative filtering

def predict_Top_K_no_Bias(ratings, similarity, k=40):
    pred = np.zeros(ratings.shape)
    user_bias = ratings.mean(axis=1)
    ratings = (ratings - user_bias[:, np.newaxis]).copy()
    for i in range(ratings.shape[0]):
        top_K_users = [np.argsort(similarity[:,i])[:-k-1:-1]]
        for j in range(ratings.shape[1]):
            pred[i,j] = similarity[i, :][top_K_users].dot(ratings[:, j][top_K_users])
            pred[i,j] /= np.sum(np.abs(similarity[i, :][top_K_users]))
    pred += user_bias[:, np.newaxis]
    return pred

#----------------------------------------------------------------------
#----------------------------------------------------------------------

#BASIC SCRIPT#----------------------------------------------------------------------

newUserID = 944  # new user's id
timestamp = '883446543'  # random timestamp, we don't care about that
known_positives = []
mySelMovies = []

#read the moviset
ids_titles = readMovieSet('u.item')
#read the full dataset
fullDataSet = readFullDataset('u.data')

#insert new user
fullDataSetNewUser = insertNewUserRatings(ids_titles, fullDataSet, newUserID, timestamp, known_positives, mySelMovies)

#calculate number of users and items
n_users = numberOfUsers(fullDataSetNewUser)
 n_items = numberOfMovies(fullDataSetNewUser)

#calculate user item matrix
user_item_matrix = getUserItemMatrix(n_users, n_items, fullDataSetNewUser)

#calculate user similarity(Pearson correlation)
user_similarityPearson = calculateUsersPearsonCorrelation(user_item_matrix)

#apply bias subtracted user-based collaborative filtering with Top-40 most common neighbors algorithm
user_prediction_User = predict_Top_K_no_Bias(user_item_matrix, user_similarityPearson, k=40)
#function for printing the top n recommended movies for a given user id -
def printPredictedMoviesUserBased(user, n):
    user = user - 1
    n = n - 1
    pred_indexes = [i + 1 for i in np.argsort(-user_prediction_User[user])]
    pred_indexes = [item for item in pred_indexes if item not in known_positives]
    movies_ids_titles = pd.read_csv('u.item', sep='|', header=None, encoding='latin-1', names=['itemId', 'title'], usecols=[0, 1])
    pd_pred_indexes = pd.DataFrame(pred_indexes, columns=['itemId'])
    pred_movies = pd.merge(pd_pred_indexes, movies_ids_titles, on='itemId')
    print('
')
    print('**user-based collaborative filtering (Top-K neighbors and Bias-subtracted)**
    print(pred_movies.loc[:n])

#print the top 10 recommended movies for the new User (id = 944)
printPredictedMoviesUserBased(944, 10)
demographic-based.py

```python
import zipfile
from numpy import array
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import pairwise_distances
import math

#fetch demographic data

def _read_raw_data(path):
    with zipfile.ZipFile(path) as datafile:
        return datafile.read('ml-100k/u.user').decode(errors='ignore').split('\n')

# create the user_meta-data list

def createUserMetaDataList(users_raw, users_age, users_occup, user_meta_raw):
    # first create the user_meta-data list by the existing dataset
    for line in users_raw:
        if not line:
            continue
        #print(line)
        split = line.split('|')
        # Zero-based indexing
        userid = int(split[0])
        age = int(split[1])
        gender = split[2]
        occup = split[3]
        i = 0
        for m in users_age:
            if (age <= int(m)):
                #print(i)
                break
            else:
                i = i + 1
        if (gender == 'M'):
            j = 8
        else:
            j = 9
        k = 10
        for n in users_occup:
            if (occup == n):
                #print(k)
                break
            else:
                k = k + 1
        s = str(userid) + "|"
        for l in range(0, 31):
```

-73-
if(l == i or l == j or l == k):
    s = s + "1"
else:
    s = s + "0"

user_meta_raw.append(s)

# then, append the new user to the above user_meta_data list

while (True):
    try:
        print('Select your age range:
          '1. <=18
          '2. 19-24
          '3. 25-30
          '4. 31-40
          '5. 41-50
          '6. 51-60
          '7. 61-70
          '8. 71-100')
        newage = int(input("Choose the corresponding number: "))
    except ValueError:
        print("Wrong input, please insert an integer")
        continue
    if (newage < 1 or newage > 8):
        print("Value must be between 1 and 8. Please insert a valid integer")
        continue
    if (1 <= newage and 8 >= newage):
        newage = (newage) - 1
    break

while (True):
    try:
        print('Select your gender:
          '1. Male
          '2. Female')
        newgend = int(input("Choose the corresponding number: "))
    except ValueError:
        print("Wrong input, please insert an integer")
        continue
    if (newgend < 1 or newgend > 2):
        print("Value must be 1 or 2. Please insert a valid integer")
        continue
    if (1 == newgend or 2 == newgend):
        newgend = (newgend) - 1 + 8
    break

while (True):
    try:
        print('Select your occupation:
          '1. administra-
          'tor
          '2. artist
          '3. doctor
          '4. educator
          '5. engineer
          '6. entertainer
          '7. executive
          '8. healthcare
          '9. homemaker
          '10. lawyer
          '11. librarian
          '12. marketing
          '13. none
          '14. other
          '15. programmer
          '16. retired
          '17. salesman
          '18. scientist
          '19. student
          '20. technician
          '21. writer')
        newoccup = int(input("Choose the corresponding num-ber: "))
    except ValueError:
        print("Wrong input, please insert an integer")
        continue
    if (newoccup < 1 or newoccup > 21):
        print("Value must be between 1 and 21. Please insert a valid integer")
```python
    continue
    if (1 <= newoccup and 21 >= newoccup):
        newoccup = (newoccup) - 1 + 10
    break

    s = str(944) + "|
    for l in range(0, 31):
        if (l == newage or l == newgend or l == newoccup):
            s = s + "1|
        else:
            s = s + "0"
    s = s[:-1]
    user_meta_raw.append(s)
    return

#--------------------------------------------------
###transform users metadata to a list with zeros and ones
def _parse_user_metadata(num_users, user_meta_raw, users_combined_features):
    user_features = np.zeros((num_users, len(users_combined_features)))
    for meta in user_meta_raw:
        if not meta:
            continue
        splt = meta.split('|')
        # Zero-based indexing
        iid = int(splt[0]) - 1
        item_meta = [idx for idx, val in enumerate(splt[1:])
                     if int(val) > 0]
        for gid in item_meta:
            user_features[iid, gid] = 1.0
    return user_features

#--------------------------------------------------
###calculate the euclidean distance of users and then find the k(k=20) most common neighbors based on demographics
def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):
    distances = []
    length = len(testInstance)
    for x in range(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distances.append(dist)
    a = array(distances)
```
sorted_indexes = np.argsort(a)
index_neighbors = sorted_indexes[:k]
return index_neighbors

#We read in the u.data file, which contains the full dataset.
def readFullDataset(dataSetFilePath):
    header = ['user_id', 'item_id', 'rating', 'timestamp']
    return pd.read_csv(dataSetFilePath, sep='\t', names=header)

#we read the the movies titles from the movie dataset
def readMovieSet(movieSetFilePath):
df_ids_titles = pd.read_csv(movieSetFilePath, sep='|',
header=None, encoding='latin-1', names=['itemId', 'title'], usecols=[0, 1])
ids_titles = np.empty(1682, dtype=np.object)
for line in df_ids_titles.itertuples():
    ids_titles[line[0]] = line[2]
return ids_titles

#we count the number of unique users and movies.
def numberOfUsers(fullDataSet):
    n_users = fullDataSet.user_id.unique().shape[0]
    return n_users

def numberOfMovies(fullDataSet):
    n_items = fullDataSet.item_id.unique().shape[0]
    return n_items

#we create user-item matrix with the k most common users based on demographics
def getUserItemMatrixDemographicsBased(n_users, n_items, fullDataSet, neighbors):
    user_item_matrixTrain = np.zeros((n_users, n_items))
    for line in fullDataSet.itertuples():
        if line[1] - 1 in neighbors:
neighborsmovies = []
for i in range(0, n_items):
    for u in neighbors:
        if ((user_item_matrixTrain[u, i] == 5) and (i not in neighborsmovies)):
            return neighborsmovies.append(i)

# insert new user by creating gui in python console

def insertNewUserRatings(ids_titles, fullDataSet, newUserID, timestamp, known_positives, mySelMovies, neighborsmovies):

    i=0
    j=20
    f=0
    while(f < 10):
        userList = []
        for x in range(i,j):
            userList.append({x%20+1: ids_titles[neighborsmovies[x]])
        print('
')
        for p in userList:
            print(p)
        print('
')
        while(True):
            try:
                var = int(input("Choose a movie, or press -1 to change movieset: "))
            except ValueError:
                print("Wrong input, please insert an integer")
                continue
            if((var<-1 or var>20) and var ==0):
                print("Value must be -1 OR between 1 and 20. Please insert a valid integer")
                continue
            if(1<=var and 20>var):
                selMovie = str(ids_titles.tolist().index(userList[var-1][var])) + 1
                if selMovie in mySelMovies:
                    print("You have already selected that movie, please choose another movie")
                    continue
                mySelMovies.append(str(ids_titles.tolist().index(userList[var-1][var]))+1))
            break
            if (var == -1):
                if ((len(neighborsmovies)-1 - j) >= 20):
                    i = j
                    j += 20
                elif ((len(neighborsmovies)-1 - j) > 0):
                    i = j
                    j = len(neighborsmovies)-1
                else:
                    i = 0
                    j = 20
```python
    continue
else:
    print('\n')
    print("You selected the movie: " + userList[var-1][var] + " with ID: " + str(ids_titles.tolist().index(userList[var-1][var])+1))
    print('\n')
    while (True):
        try:
            rating = int(input("Rate the movie: "))
        except ValueError:
            print("Wrong input, please insert an integer")
            continue
        if (rating < 1 or rating > 5):
            print("Value must be between 1 and 5. Please insert a valid integer")
            continue
        break
    known_positives.append(ids_titles.tolist().index(userList[var-1][var]) + 1)
    fullDataSet.loc[len(fullDataSet)] = [newUserID, ids_titles.tolist().index(userList[var-1][var]) + 1, rating, timestamp]
    f = f + 1
    while(f < 10):
        while (True):
            try:
                ch = int(input("To change the movieset press -1, to keep press 1: "))
            except ValueError:
                print("Wrong input, please insert an integer")
                continue
            if (ch != 1 and ch != -1):
                print("Value must be 1 or -1. Please insert a valid integer")
                continue
            break
    if(int(ch) == -1):
        break
    else:
        print('\n')
        for p in userList:
            print(p)
        print('\n')
        while (True):
            try:
                var = int(input("Choose a movie: "))
            except ValueError:
                print("Wrong input, please insert an integer")
                continue
    if ((var < -1 or var > 20)):
        print("Value must be between 1 and 20. Please insert a valid integer")
        continue
    selMovie = str(ids_titles.tolist().index(userList[var-1][var]) + 1)
    if selMovie in mySelMovies:
```
print("You have already selected that
movie, please choose another movie")
continue
mySelMovies.append(str(ids_titles.tolist().index(userList[var - 1][var]) + 1))
break
print('
"
print("You selected the movie: " + us-
erList[var - 1][var] + " with ID: " + str(
ids_titles.tolist().index(userList[var - 1][var]) + 1))
print('
"
while (True):
    try:
        rating = int(input("Rate the movie:
"))
    except ValueError:
        print("Wrong input, please insert an
integer")
        continue
    if (rating < 1 or rating > 5):
        print("Value must be between 1 and 5.
Please insert a valid integer")
        continue
    break
    known_positives.append(ids_titles.tolist().index(userList[var - 1][var]) + 1)
    fullDataSet.loc[len(fullDataSet)] = [newUserID, ids_titles.tolist().index(userList[var - 1][var]) + 1, rating, timestamp]
f = f + 1
if((len(neighborsmovies)-1-j) >= 20):
    i = j
    j += 20
elif((len(neighborsmovies)-1-j) > 0):
    i = j
    j = len(neighborsmovies)-1
else:
    i = 0
    j = 20
print('
"
return fullDataSet

#we create user-item matrix
def getUserItemMatrix(n_users, n_items, fullDataSet):
    user_item_matrix = np.zeros((n_users, n_items))
    for line in fullDataSet.itertuples():
    return user_item_matrix

#---
we use the pairwise_distances function from sklearn to calculate the pearson correlation

def calculateUsersPearsonCorrelation(user_item_matrixTrain):
    user_similarityPearson = 1 - pairwise_distances(user_item_matrixTrain, metric='correlation') #943*943
    user_similarityPearson[np.isnan(user_similarityPearson)] = 0
    return user_similarityPearson

#make predictions combining Top-K neighbors and Bias-subtracted collaborative filtering

def predict_Top_K_no_Bias(ratings, similarity, k=40):
    pred = np.zeros(ratings.shape)
    user_bias = ratings.mean(axis=1)
    ratings = (ratings - user_bias[:, np.newaxis]).copy()
    for i in range(ratings.shape[0]):
        top_K_users = [np.argsort(similarity[:,i])[:-k-1:-1]]
        for j in range(ratings.shape[1]):
            pred[i,j] = similarity[i, :][top_K_users].dot(ratings[:, j][top_K_users])
            pred[i,j] /= np.sum(np.abs(similarity[i, :][top_K_users]))
    pred += user_bias[:, np.newaxis]
    return pred

#fetch demographic data
users_raw = _read_raw_data("C:/Users/Sak/lightfm_data/movielens100k/movielens.zip")

# create models
users_age = ['18', '24', '30', '40', '50', '61', '70', '100']
users_occup = ['administrator', 'artist', 'doctor', 'educator', 'engineer', 'entertainer', 'executive', 'healthcare', 'homemaker', 'lawyer', 'librarian', 'marketing', 'none', 'other', 'programmer',...
users_combined_features = ['18|0', '24|1', '30|2', '40|3', '50|4', '61|5', '70|6', '100|7', 'm|8', 'f|9', 'administrator|10', 'artist|11', 'doctor|12', 'educator|13', 'engineer|14', 'entertainer|15', 'executive|16', 'healthcare|17', 'homemaker|18', 'lawyer|19', 'librarian|20', 'marketing|21', 'none|22', 'other|23', 'programmer|24', 'retired|25', 'salesman|26', 'scientist|27', 'student|28', 'technician|29', 'writer|30']

user_meta_raw = []

user_meta_raw = createUserMetaDataList(users_raw, users_age, users_occup, user_meta_raw)

#read the movieset
ids_titles = readMovieSet('u.item')

#read the full dataset
fullDataSet = readFullDataset('u.data')

#users demographic data with new user
usr_feat = _parse_user_metadata(944, user_meta_raw, users_combined_features)

#users demographic data without the new user
usr_feat_no_newUser = np.delete(usr_feat, (943), axis=0)

#new user demographic data
new_usr_feat = usr_feat[-1]

#The 20 most common neighbors for the new user based on demographics are:
neighbors = getNeighbors(usr_feat_no_newUser, new_usr_feat, 10)

#calculate number of users and items
n_users = numberOfUsers(fullDataSet)

n_items = numberOfMovies(fullDataSet)

#neighbors movies the k most common users based on demographics
neighborsMovies = getUserItemMatrixDemographicsBased(n_users, n_items, fullDataSet, neighbors)

#full dataset with new users ratings
fullDataSetNewUser = insertNewUserRatings(ids_titles, fullDataSet, newUserID, timestamp, known_positives, mySelMovies, neighborsMovies)

#calculate number of users and items
n_users = numberOfUsers(fullDataSetNewUser)

n_items = numberOfMovies(fullDataSetNewUser)

#calculate user item matrix with new dataset
user_item_matrix = getUserItemMatrix(n_users, n_items, fullDataSetNewUser)

#calculate user similarity(Pearson correlation)
user_similarityPearson = calculateUsersPearsonCorrelation(user_item_matrix)

# apply bias subtracted user-based collaborative filtering with Top-40 most common neighbors algorithm
user_prediction_User = predict_Top_K_no_Bias(user_item_matrix, user_similarityPearson, k=40)

# function for printing the top n recommended movies for a given user id -
def printPredictedMoviesUserBased(user, n):
    user = user - 1
    n = n - 1
    pred_indexes = [i + 1 for i in np.argsort(-user_prediction_User[user])]
    pred_indexes = [item for item in pred_indexes if item not in known_positives]
    movies_ids_titles = pd.read_csv('u.item', sep='|', header=None, encoding='latin-1', names=['itemId', 'title'], usecols=[0, 1])
    pd_pred_indexes = pd.DataFrame(pred_indexes, columns=['itemId'])
    pred_movies = pd.merge(pd_pred_indexes, movies_ids_titles, on='itemId')
    print('
')
    print("***********************user-based collaborative filtering (Top-K neighbors and Bias-subtracted)**************************")
    print(pred_movies.loc[:n])

# print the top 10 recommended movies for the new User (id = 944)
printPredictedMoviesUserBased(944, 10)
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import pairwise_distances
import math

# We read in the u.data file, which contains the full dataset.
def readFullDataset(dataSetFilePath):
    header = ['user_id', 'item_id', 'rating', 'timestamp']
    return pd.read_csv(dataSetFilePath, sep='\t', names=header)

# We read the movie titles from the movie dataset
def readMovieSet(movieSetFilePath):
    df_ids_titles = pd.read_csv(movieSetFilePath, sep='|',
                                header=None, encoding='latin-1',
                                names=['itemId', 'title'], usecols=[0, 1])
    ids_titles = np.empty(1682, dtype=np.object)
    for line in df_ids_titles.itertuples():
        ids_titles[line[0]] = line[2]
    return ids_titles

# We count the number of unique users and movies.
def numberOfUsers(fullDataSet):
    n_users = fullDataSet.user_id.unique().shape[0]
    return n_users

def numberOfMovies(fullDataSet):
    n_items = fullDataSet.item_id.unique().shape[0]
    return n_items

# Calculate existing movies entropy0 values
def calcMoviesEntropy0(fullDataSet, n_users, n_items):
    user_item_matrixTrain = np.zeros((n_users, n_items))
    for line in fullDataSet.itertuples():
values = np.zeros((n_items, 6))

for i in range(0, n_items):
    for u in range(0, n_users):
        for j in range(0, 6):
            if user_item_matrixTrain[u, i] == j:
                values[i, j] += 1

voters = np.zeros((n_items))

for i in range(0, n_items):
    voters[i] = values[i, 1] + values[i, 2] + values[i, 3] + values[i, 4] + values[i, 5]

prop = np.zeros((n_items, 6))

w = np.zeros(6)

for i in range(0, 6):
    if i == 0:
        w[i] = 0.5
    else:
        w[i] = 1

for i in range(0, n_items):
    prop[i, 0] = values[i, 0]/voters[i]
    prop[i, 1] = values[i, 1]/voters[i]
    prop[i, 2] = values[i, 2]/voters[i]
    prop[i, 3] = values[i, 3]/voters[i]
    prop[i, 4] = values[i, 4]/voters[i]
    prop[i, 5] = values[i, 5]/voters[i]

entropy = np.zeros((n_items))

for i in range(0, n_items):
    entropy[i] = 0
    for rat in range(0, 6):
        if prop[i, rat] != 0:
            entropy[i] = entropy[i] + prop[i, rat]*w[rat]*math.log(prop[i, rat], 2)
    entropy[i] = entropy[i]/5.5

entropy = -entropy

entropy_indexes = [i for i in np.argsort(-entropy)]

#insert new user by creating gui in python console

def insertNewUserRatings(ids_titles, fullDataSet, newUserID, timestamp, known_positives, mySelMovies, entropy_indexes):
    i=0
    j=20
    f=0
    while(f < 10):
        userList = []
        for x in range(i,j):
            userList.append({x%20+1: ids_titles[entropy_indexes[x]]})


```python
print('n')
for p in userList:
    print(p)
print('n')
while(True):
    try:
        var = int(input("Choose a movie, or press -1 to change movieset: "))
    except ValueError:
        print("Wrong input, please insert an integer")
        continue
    if((var<-1 or var>20) and var ==0):
        print("Value must be -1 OR between 1 and 20.
Please insert a valid integer")
        continue
    selMovie = str(ids_titles.tolist().index(userList[var - 1][var]) + 1)
    if selMovie in mySelMovies:
        print("You have already selected that movie, please choose another movie")
        continue
    mySelMovies.append(str(ids_titles.tolist().index(userList[var - 1][var]) + 1))
    break
    elif (var == -1):
        if (1681 - j) >= 20:
            i = j
            j += 20
        elif (1681 - j) > 0:
            i = j
            j = 1682
        else:
            i = 0
            j = 20
        continue
    else:
        print('n')
        print("You selected the movie: " + userList[var - 1][var] + " with ID: " + str(ids_titles.tolist().index(userList[var - 1][var]) + 1))
        print('n')
        while (True):
            try:
                rating = int(input("Rate the movie: "))
            except ValueError:
                print("Wrong input, please insert an integer")
                continue
            if (rating < 1 or rating > 5):
                print("Value must be between 1 and 5. Please insert a valid integer")
                continue
            known_positives.append(ids_titles.tolist().index(userList[var - 1][var]) + 1)
            fullDataSet.loc[len(fullDataSet)] = [newUserID, ids_titles.tolist().index(userList[var - 1][var]) + 1, rating, timestamp]
            f = f + 1
            while(f < 10):
```

-85-
while (True):
    try:
        ch = int(input("To change the movieset press -1, to keep press 1: "))
    except ValueError:
        print("Wrong input, please insert an integer")
        continue
    if (ch != 1 and ch != -1):
        print("Value must be 1 or -1. Please insert a valid integer")
        continue
    break
    if(int(ch) == -1):
        break
    else:
        print('
')
        for p in userList:
            print(p)
        print('
')
        while (True):
            try:
                var = int(input("Choose a movie: "))
            except ValueError:
                print("Wrong input, please insert an integer")
                continue
            if ((var < -1 or var > 20)):
                print("Value must be between 1 and 20. Please choose another movie")
                continue
            selMovie = str(ids_titles.tolist().index(userList[var - 1][var]) + 1)
            if selMovie in mySelMovies:
                print("You have already selected that movie, please choose another movie")
            else:
                if selMovie in known_positives:
                    print("You have already rated this movie, please choose another movie")
                else:
                    print("You selected the movie: " + userList[var - 1][var] + " with ID: " + str(ids_titles.tolist().index(userList[var - 1][var]) + 1))
                    print('
')
                    while (True):
                        try:
                            rating = int(input("Rate the movie: "))
                        except ValueError:
                            print("Wrong input, please insert an integer")
                            continue
                        if (rating < 1 or rating > 5):
                            print("Value must be between 1 and 5. Please insert a valid integer")
                        else:
                            known_positives.append(ids_titles.tolist().index(userList[var - 1][var]) + 1)
fullDataSet.loc[len(fullDataSet)] = [newUserID, ids_titles.tolist().index(userList[var - 1][var]) + 1, rating, timestamp]

    f = f + 1
    if ((1681-j) >= 20):
        i = j
        j += 20
    elif ((1681-j) > 0):
        i = j
        j = 1682
    else:
        i = 0
        j = 20
    print('
')
    return fullDataSet

#we create user-item matrix

def getUserItemMatrix(n_users, n_items, fullDataSet):
    user_item_matrix = np.zeros((n_users, n_items))
    for line in fullDataSet.itertuples():
        user_item_matrix[line[1]-1, line[2]-1] = line[3]
    return user_item_matrix

#we use the pairwise_distances function from sklearn to calculate the pearson correlation

def calculateUsersPearsonCorrelation(user_item_matrixTrain):
    user_similarityPearson = 1 - pairwise_distances(user_item_matrixTrain, metric='correlation') #943*943
    user_similarityPearson[np.isnan(user_similarityPearson)] = 0
    return user_similarityPearson

#make predictions combining Top-K neighbors and Bias-subtracted collaborative filtering

def predict_Top_K_no_Bias(ratings, similarity, k=40):
    pred = np.zeros(ratings.shape)
    user_bias = ratings.mean(axis=1)
    ratings = (ratings - user_bias[:, np.newaxis]).copy()
    for i in range(ratings.shape[0]):
        top_K_users = [np.argsort(similarity[:,i])[-k-1:-1]]
        for j in range(ratings.shape[1]):
            pred[i,j] = similarity[i, :][top_K_users].dot(ratings[:, j][top_K_users])
pred[i,j] /= np.sum(np.abs(similarity[i,:][top_K_users]))

pred += user_bias[:, np.newaxis]

return pred

#---------------------

# BASIC SCRIPT

newUserID = 944  # new user's id
timestamp = '883446543'  # random timestamp, we don't care about that
known_positives = []
mySelMovies = []

# read the movieset
ids_titles = readMovieSet('u.item')

# read the full dataset
fullDataSet = readFullDataset('u.data')

# calculate number of users and items
n_users = numberOfUsers(fullDataSet)
n_items = numberOfMovies(fullDataSet)

# calculate movies ratings entropy0 values and return movies indexes with the highest entropy0 values to the lowest
entropy_indexes = calcMoviesEntropy0(fullDataSet, n_users, n_items)

# insert new user
fullDataSetNewUser = insertNewUserRatings(ids_titles, fullDataSet, newUserID, timestamp, known_positives, mySelMovies, entropy_indexes)

# calculate number of users and items with new user
n_users = numberOfUsers(fullDataSetNewUser)
n_items = numberOfMovies(fullDataSetNewUser)

# calculate user item matrix
user_item_matrix = getUserItemMatrix(n_users, n_items, fullDataSetNewUser)

# calculate user similarity (Pearson correlation)
user_similarityPearson = calculateUsersPearsonCorrelation(user_item_matrix)

# apply bias subtracted user-based collaborative filtering with Top-40 most common neighbors algorithm
user_prediction_User = predict_Top_K_no_Bias(user_item_matrix, user_similarityPearson, k=40)

# function for printing the top n recommended movies for a given user id

def printPredictedMoviesUserBased(user, n):
    user = user - 1
n = n - 1
pred_indexes = [i + 1 for i in np.argsort(-user_prediction_User[user])]
pred_indexes = [item for item in pred_indexes if item not in known_positives]
movies_ids_titles = pd.read_csv('u.item', sep="|", header=None, encoding='latin-1', names=['itemId', 'title'], usecols=[0, 1])
pd_pred_indexes = pd.DataFrame(pred_indexes, columns=['itemId'])
pred_movies = pd.merge(pd_pred_indexes, movies_ids_titles, on='itemId')
print('
')
print("***************user-based collaborative filtering (Top-K neighbors and Bias-subtracted)***************")
print(pred_movies.loc[:n])

# print the top 10 recommended movies for the new User (id = 944)
printPredictedMoviesUserBased(944, 10)
import zipfile
from numpy import array
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import pairwise_distances
import math

######################################################################
##################################################
# fetch demographic data
def _read_raw_data(path):
    with zipfile.ZipFile(path) as datafile:
        return datafile.read('ml-100k/u.user').decode(errors='ignore').split('
')

#--=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-#

######################################################################
#############################################
#####
# create the user_meta-data list
def createUserMetaDataList(users_raw, users_age, users_occup, user_meta_raw):
    # first create the user_meta-data list by the existing dataset
    for line in users_raw:
        if not line:
            continue
        #print(line)
        split = line.split('|')
        # Zero-based indexing
        userid = int(split[0])
        age = int(split[1])
        gender = split[2]
        occup = split[3]
        i = 0
        for m in users_age:
            if (age <= int(m)):
                #print(i)
                break
            else:
                i = i + 1
        if(gender == 'M'):
            j = 8
        else:
            j = 9
        k = 10
        for n in users_occup:
            if(occup == n):
                #print(k)
                break
            else:
                k = k + 1
        s = str(userid) + '|' +
        for l in range(0, 31):
if(l == i or l == j or l == k):
    s = s + "1|
else:
    s = s + "0|

user_meta_raw.append(s)

# then, append the new user to the above user_meta_data list

while (True):
    try:
        print('Select your age range:
', '1. <=18\n', '2. 19-24\n', '3. 25-30\n', '4. 31-40\n', '5. 41-50\n',
              '6. 51-60\n', '7. 61-70\n', '8. 71-100\n')
        newage = int(input("Choose the corresponding number: "))
    except ValueError:
        print("Wrong input, please insert an integer")
        continue
    if (newage < 1 or newage > 8):
        print("Value must be between 1 and 8. Please insert a valid integer")
    continue
    if (1 <= newage and 8 >= newage):
        newage = (newage) - 1
    break

while (True):
    try:
        print('Select your gender:
', '1. Male\n', '2. Female\n')
        newgend = int(input("Choose the corresponding number: "))
    except ValueError:
        print("Wrong input, please insert an integer")
        continue
    if (newgend < 1 or newgend > 2):
        print("Value must be 1 or 2. Please insert a valid integer")
    continue
    if (1 == newgend or 2 == newgend):
        newgend = (newgend) - 1 + 8
    break

while (True):
    try:
        print('Select your occupation:
', '1. administrator\n', '2. artist\n', '3. doctor\n', '4. educator\n',
              '5. engineer\n', '6. entertainer\n',
              '7. executive\n', '8. healthcare\n', '9. homemaker\n', '10. lawyer\n', '11. librarian\n',
              '12. marketing\n', '13. none\n', '14. other\n',
              '15. programmer\n', '16. retired\n', '17. salesman\n', '18. scientist\n', '19. student\n',
              '20. technician\n', '21. writer\n')
        newoccup = int(input("Choose the corresponding number: "))
    except ValueError:
        print("Wrong input, please insert an integer")
        continue
    if (newoccup < 1 or newoccup > 21):
        print("Value must be between 1 and 21. Please insert a valid integer")

-91-
continue
    if (1 <= newoccup and 21 >= newoccup):
        newoccup = (newoccup) - 1 + 10
    break

s = str(944) + "|"
for l in range(0, 31):
    if (l == newage or l == newgend or l == newoccup):
        s = s + "1|
    else:
        s = s + "0|
s = s[:-1]
user_meta_raw.append(s)
return user_meta_raw

#transform users metadata to a list with zeros and ones
def _parse_user_metadata(num_users, user_meta_raw, users_combined_features):
    user_features = np.zeros((num_users, len(users_combined_features)))
    for meta in user_meta_raw:
        if not meta:
            continue
        splt = meta.split('|')
        # Zero-based indexing
        iid = int(splt[0]) - 1
        item_meta = [idx for idx, val in enumerate(splt[1:])
            if int(val) > 0]
        for gid in item_meta:
            user_features[iid, gid] = 1.0
    return user_features

#calculate the euclidean distance of users and then find the
#k(k=20) most common neighbors based on demographics
def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)
def getNeighbors(trainingSet, testInstance, k):
    distances = []
    length = len(testInstance)
    for x in range(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distances.append(dist)
a = array(distances)
sorted_indexes = np.argsort(a)
index_neighbors = sorted_indexes[:k]
return index_neighbors

# We read in the u.data file, which contains the full dataset.
def readFullDataset(dataSetFilePath):
    header = ['user_id', 'item_id', 'rating', 'timestamp']
    return pd.read_csv(dataSetFilePath, sep='\t', names=header)

# we read the the movies titles from the movie dataset
def readMovieSet(movieSetFilePath):
    df_ids_titles = pd.read_csv(movieSetFilePath, sep='\|',
        header=None, encoding='latin-1', names=['itemId', 'title'], usecols=[0, 1])
    ids_titles = np.empty(1682, dtype=np.object)
    for line in df_ids_titles.itertuples():
        ids_titles[line[0]] = line[2]
    return ids_titles

# we count the number of unique users and movies.
def numberOfUsers(fullDataSet):
    n_users = fullDataSet.user_id.unique().shape[0]
    return n_users

def numberOfMovies(fullDataSet):
    n_items = fullDataSet.item_id.unique().shape[0]
    return n_items

def calcMoviesEntropy0(fullDataSet, n_users, n_items, neighbors):
    user_item_matrixTrain = np.zeros((n_users, n_items))
    for line in fullDataSet.itertuples():
        if line[1] - 1 in neighbors:

values = np.zeros((n_items, 6))

for i in range(0, n_items):
    for u in range(0, n_users):
        for j in range(0, 6):
            if user_item_matrixTrain[u, i] == j:
                values[i, j] += 1

voters = np.zeros((n_items))

for i in range(0, n_items):
    voters[i] = values[i, 1] + values[i, 2] + values[i, 3] + values[i, 4] + values[i, 5]

prop = np.zeros((n_items, 6))

w = np.zeros(6)

for i in range(0, 6):
    if i == 0:
        w[i] = 0.5
    else:
        w[i] = 1

for i in range(0, n_items):
    if voters[i] != 0:
        prop[i, 0] = values[i, 0]/voters[i]
        prop[i, 1] = values[i, 1]/voters[i]
        prop[i, 2] = values[i, 2]/voters[i]
        prop[i, 3] = values[i, 3]/voters[i]
        prop[i, 4] = values[i, 4]/voters[i]
        prop[i, 5] = values[i, 5]/voters[i]

entropy = np.zeros((n_items))

for i in range(0, n_items):
    entropy[i] = 0
    for rat in range(0, 6):
        if prop[i, rat] != 0:
            entropy[i] = entropy[i] + prop[i, rat]*w[rat]*math.log(prop[i, rat], 2)
    entropy[i] = entropy[i]/5.5

entropy = -entropy

entropy_indexes = [i for i in np.argsort(-entropy)]

return entropy_indexes

#insert new user by creating gui in python console

def insertNewUserRatings(ids_titles, fullDataSet, newUserID, timestamp, known_positives, mySelMovies, entropy_indexes):
    i = 0
    j = 20
    f = 0
    while (f < 10):
userList = []
for x in range(i, j):
    userList.append({x%20+1: ids_titles[entropy_indexes[x]]})
    print('
')
for p in userList:
    print(p)
print('
')
while(True):
    try:
        var = int(input("Choose a movie, or press -1 to change movieset: "))
    except ValueError:
        print("Wrong input, please insert an integer")
        continue
    if((var<-1 or var>20) and var ==0):
        print("Value must be -1 OR between 1 and 20. Please insert a valid integer")
        continue
    if(1<=var and 20>=var):
        selMovie = str(ids_titles.tolist().index(userList[var - 1][var]) + 1)
        if selMovie in mySelMovies:
            print("You have already selected that movie, please choose another movie")
            continue
        mySelMovies.append(str(ids_titles.tolist().index(userList[var - 1][var]) + 1))
        break
    if (var == -1):
        if ((1681 - j) >= 20):
            i = j
        elif ((1681 - j) > 0):
            i = j
            j = 1682
        else:
            i = 0
            j = 20
        continue
    else:
        print('
')
        print("You selected the movie: " + userList[var-1][var] + " with ID: " + str(ids_titles.tolist().index(userList[var-1][var])+1))
        print('
')
        while (True):
            try:
                rating = int(input("Rate the movie: "))
            except ValueError:
                print("Wrong input, please insert an integer")
                continue
            if (rating < 1 or rating > 5):
                print("Value must be between 1 and 5. Please insert a valid integer")
                continue
            break
            known_positives.append(ids_titles.tolist().index(userList[var - 1][var]) + 1)
```python
fullDataSet.loc[len(fullDataSet)] = [newUserID, ids_titles.tolist().index(userList[var - 1][var]) + 1, rating, timestamp]
f = f + 1
while (f < 10):
    try:
        ch = int(input("To change the movieset press -1, to keep press 1: "))
    except ValueError:
        print("Wrong input, please insert an integer")
    if (ch != 1 and ch != -1):
        print("Value must be 1 or -1. Please insert a valid integer")
        continue
        break
    if (int(ch) == -1):
        break
    else:
        print('
')
        for p in userList:
            print(p)
        print('
')
        while (True):
            try:
                var = int(input("Choose a movie: "))
            except ValueError:
                print("Wrong input, please insert an integer")
                continue
                if ((var < -1 or var > 20)):
                    print("Value must be between 1 and 20. Please insert a valid integer")
                    continue
                    selMovie = str(ids_titles.tolist().index(userList[var - 1][var]) + 1)
                    if selMovie in mySelMovies:
                        print("You have already selected that movie, please choose another movie")
                        continue
                        mySelMovies.append(str(ids_titles.tolist().index(userList[var - 1][var]) + 1))
                        break
                        print("You selected the movie: " + userList[var - 1][var] + " with ID: " + str(ids_titles.tolist().index(userList[var - 1][var]) + 1))
                        print('
')
                        while (True):
                            try:
                                rating = int(input("Rate the movie: "))
                            except ValueError:
                                print("Wrong input, please insert an integer")
                                continue
                                if (rating < 1 or rating > 5):
```

Please insert a valid integer")
continue
break
known_positives.append(ids_titles.tolist().index(userList[var - 1][var]) + 1)
fullDataSet.loc[len(fullDataSet)] = [newUserID, ids_titles.tolist().index(userList[var - 1][var]) + 1, rating, timestamp]
f = f + 1
if((1681-j) >= 20):
i = j
j += 20
elif((1681-j) > 0):
i = j
j = 1682
else:
i = 0
j = 20
print('
')
return fullDataSet
#we create user-item matrix
def getUserItemMatrix(n_users, n_items, fullDataSet):
user_item_matrix = np.zeros((n_users, n_items))
for line in fullDataSet.itertuples():
return user_item_matrix
#we use the pairwise_distances function from sklearn to calculate the pearson correlation
def calculateUsersPearsonCorrelation(user_item_matrixTrain):
user_similarityPearson = 1 - pairwise_distances(user_item_matrixTrain, metric='correlation') #943*943
user_similarityPearson[np.isnan(user_similarityPearson)] = 0
return user_similarityPearson
#make predictions combining Top-K neighbors and Bias-subtracted collaborative filtering
def predict_Top_K_no_Bias(ratings, similarity, k=40):
pred = np.zeros(ratings.shape)
user_bias = ratings.mean(axis=1)
ratings = (ratings - user_bias[:,:].copy())

for i in range(ratings.shape[0]):
    top_K_users = [np.argsort(similarity[:,i])[:k-1:-1]]
    for j in range(ratings.shape[1]):
        pred[i,j] = similarity[i, :][top_K_users].dot(ratings[:, j][top_K_users])
        pred[i,j] /= np.sum(np.abs(similarity[i, :][top_K_users]))
    pred += user_bias[:, np.newaxis]
return pred

#---------------------------------------------------

newUserID = 944  # new user's id
timestamp = '883446543'  # random timestamp, we don't care about that
known_positives = []
mySelMovies = []

#fetch dempgraphic data
users_raw = _read_raw_data("C:/Users/Sak/lightfm_data/movielens100k/movielens.zip")

# create models
users_age = ['18', '24', '30', '40', '50', '61', '70', '100']
users_occup = ['administrator', 'artist', 'doctor', 'educator',
               'engineer', 'entertainer', 'executive', 'healthcare', 'homemaker',
               'lawyer', 'librarian', 'marketing',
               'none', 'other', 'programmer',
               'retired', 'salesman', 'scientist',
               'student', 'technician',
               'writer']

users_combined_features = ['18|0', '24|1', '30|2', '40|3',
                          '50|4', '61|5', '70|6', '100|7',
                          'm|8', 'f|9', 'administrator|10',
                          'artist|11', 'doctor|12', 'educator|13',
                          'engineer|14', 'entertainer|15',
                          'executive|16', 'healthcare|17',
                          'homemaker|18',
                          'lawyer|19', 'librarian|20',
                          'marketing|21', 'none|22', 'other|23',
                          'programmer|24',
                          'retired|25', 'salesman|26',
                          'scientist|27', 'student|28', 'technician|29',
                          'writer|30']

user_meta_raw = []

user_meta_raw = createUserMetaDataList(users_raw, users_age, users_occup, user_meta_raw)

# users demographic data with new user
usr_feat = _parse_user_metadata(944, user_meta_raw, users_combined_features)

# Users demographic data without the new user
usr_feat_no_newUser = np.delete(usr_feat, (943), axis=0)

# New user demographic data
new_usr_feat = usr_feat[-1]

# The 20 most common neighbors for the new user based on demographics are:
neighbors = getNeighbors(usr_feat_no_newUser, new_usr_feat, 10)

# Read the movieset
ids_titles = readMovieSet('u.item')

# Read the full dataset
fullDataSet = readFullDataset('u.data')

# Calculate number of users and items
n_users = numberOfUsers(fullDataSet)

# Calculate number of movies
n_items = numberOfMovies(fullDataSet)

# Calculate only neighbors movies ratings entropy0 values and return movies indexes with the highest entropy0 values to the lowest
entropy_indexes = calcMoviesEntropy0(fullDataSet, n_users, n_items, neighbors)

# Insert new user
fullDataSetNewUser = insertNewUserRatings(ids_titles, fullDataSet, newUserID, timestamp, known_positives, mySelMovies, entropy_indexes)

# Calculate number of users and items with new user
n_users = numberOfUsers(fullDataSetNewUser)

# Calculate number of movies with new user
n_items = numberOfMovies(fullDataSetNewUser)

# Calculate user item matrix
user_item_matrix = getUserItemMatrix(n_users, n_items, fullDataSetNewUser)

# Calculate user similarity (Pearson correlation)
user_similarityPearson = calculateUsersPearsonCorrelation(user_item_matrix)

# Apply bias subtracted user-based collaborative filtering with Top-40 most common neighbors algorithm
user_prediction_User = predict_Top_K_no_Bias(user_item_matrix, user_similarityPearson, k=40)

# Function for printing the top n recommended movies for a given user id -
def printPredictedMoviesUserBased(user, n):
    user = user - 1
    n = n - 1
    pred_indexes = [i + 1 for i in np.argsort(-user_prediction_User[user])]
    pred_indexes = [item for item in pred_indexes if item not in known_positives]
    movies_ids_titles = pd.read_csv('u.item', sep='|', header=None, encoding='latin-1', names=[itemId, 'title'], usecols=[0, 1])
pd_pred_indexes = pd.DataFrame(pred_indexes, columns=['itemId'])
pred_movies = pd.merge(pd_pred_indexes, movies_ids_titles, on='itemId')
print('
')
print("*************user-based collaborative filtering (Top-K neighbors and Bias-subtracted)***********")
print(pred_movies.loc[:n])

# Print the top 10 recommended movies for the new User (id = 944)
printPredictedMoviesUserBased(944, 10)