INTERNATIONAL
HELLENIC
UNIVERSITY

THE 50/50 RECOMMENDER: PERSONALITY IN MOVIE RECOMMENDER SYSTEMS

Nalmpantis Orestis

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Information and Communication Systems

JANUARY 2017

THESSALONIKI – GREECE

THE50/50 RECOMMENDER: PERSONALITY IN MOVIE RECOMMENDER SYSTEMS

Nalmpantis Orestis

Supervisor:		Prof. C.Tjortjis
Supervising Members:	Committee	Assoc. Prof. N. Basilliades
		Assist. Prof. C.Berberidis

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Information and Communication Systems

Abstract

This dissertation was written as a part of the MSc in ICT Systems at the International Hellenic University. Its main goal is the examination of the role of human personality in Movie Recommender systems. We introduce the concept of combining collaborative techniques with a personality test so to provide more personalized movie recommendations.

Previous research has shown some efforts to incorporate personality in Recommender systems, but no actual implementation has been attempted on a software level. Using a renowned movie dataset and the Big Five Personality test, we developed a system with Python that managed to improve the normal Movie Recommendation experience by 3.62%.

The findings show that Personalization improves the user's experience even though extra effort might be demanded. With further modifications and testing, we can come to the new age of recommender systems, where personality of the user is as important as it is in real life.

I would like to thank my supervisor Dr. Christos Tjortjis for his guidance and his excellent ideas and additions to this Dissertation.

Student Name: Nalmpantis Orestis

Date:13/01/2017

Contents

1 Introduction

The last few years, the massive growth and impact of the World Wide Web had as an immediate result the handling and distribution of huge amounts of data and information. It also made it easier for the average user to access this information and use it for his own needs.

Although this may seem as a huge improvement for computer technology, it certainly came with some drawbacks. Now, all these data had to be stored somewhere to be maintained. This, in conjunction with the fact that the data kept growing and getting bigger, made it really difficult for the average user to sort and process the databases and extract useful information. In order for the user to be able to use the data and fulfill his needs, a lot of time and effort is demanded.

This is where a recommender system comes in place. In a few words, a recommender system is an assistive device that directs and guides the user in his search for useful information. This system is the mean for the user to avoid all this effort of endless hours of searching and organizing the data and information.

1.1 History Of Recommender Systems

The very first recommender system came in the late 70's which is fairly early in the history of computers. The name of the recommender was Grundy and it was a system used for a library with the main goal of suggesting novels to people who are first organized into different stereotypes. It was pretty impressive at the time it was designed as it incorporated the personalities and goals of all the distinct users before making the recommendations.

Later, in the early 90's we saw the rise of collaborative filtering which came as a solution to the huge overload in data. One of the first systems that used collaborative filtering was Tapestry which allowed users to search for items in an information domain, based on opinions of other users.

Then, Grouplens, came and introduced automated collaborative filtering recommendation systems. Grouplens main goal was to suggest interesting Usenet articles by finding similar opinions between different users. The idea is that the active user can express whether he likes a Usenet article or not and then the system predicts and recommends him articles that he will probably like based on people who share the same taste as him. This is what we call the nearest neighbor method and it's the one that we used in this dissertation.

Collaborative filtering became widely known and this increased the interest in machine learning and data mining generally. Various recommender systems were introduced such as Bellcore Movie Recommender and Ringo Music Recommender. Furthermore, during this time, recommender systems were also applied for marketing situations and became really useful for increasing sales and generally reducing the customer effort and improving his experience.

Then the most renowned recommender system came, Amazon. This system is still famous today and uses a combination of collaborative and content based filtering method with the addition of what the user is currently browsing to make the recommendations.

A big step to the study of recommender systems came in 2006, when Netflix launched their Netflix prize competition to improve their movie recommendation algorithm. Today, Netflix is considered to have one of the more advanced hybrid recommender systems.

1.2 What Is Exactly A Recommender System?

An easy definition of a Recommender system is one that implies that it is a system which has the abilities to collect and present in each user some documents (in a general sense) which belong in his field of interest. This is exactly what we call a recommendation, the collection and presentation of useful information to a specific user.

This definition has been expanded by researchers with a great example mentioned in [**1**], where they state that a recommender system is " any system that produces individualized recommendations as output or has the effect of guiding the user in a personalized way to interesting or useful objects in a large space of possible options"

Then [**2**] made it even more formal stating that " the recommendation problem can be formulated as follows: Let *C* be the set of all users and let *S* be the set of all possible items that can be recommended. Let *u* be a utility function that measures the usefulness of item *s* to user c, that is , u:C x S →R, where R is a totally ordered set (for example, nonnegative integers or real numbers within a certain range). Then for each user $c \in C$, we want to choose such items $s' \in S$ that maximizes the user's utility".

We can clearly see from the above definition that the goal of a recommender system is the choice of items with the best correlation and not just to predict the correlations between the users and all the different items.

So from all these definitions we can come to the conclusion about two significant facts regarding the recommender systems. First, personalization is an important part of a recommender system as its main focus should be the recommendation of specific products and services to a particular user and not to represent group consensus for all users. Second, the system must know some important information about the user so to be able to make recommendations. The user must be present with discrete options, including items known in advance and not randomly generated.

1.3 Recommender Systems Today

Recommender systems today became a standard for every online store. They play the part of the sales person, a sales person that knows all the data or number of products the online service offers and knows truly what you like and what you don't. This comes really handy when users have limited time and patience and are not sure what they are looking for. Users might be surprised by the fact that recommenders may even suggest things that they didn't even know they liked.

So, recommendations help online stores and services solve the problem of discovery by providing top picks for you, suggestions in the style of "If you like this, you will also like that " and "if you buy this, you will need that." They do that with the use of huge databases that includes what the users browsed, what the users bought, what and when they clicked and what they rated.

Personalization in recommender systems tends to be a new trend nowadays. It is mostly based upon the theory of human-computer interaction which in a few words says that computers will and should always work with and for humans. Lately, computer scientists try to incorporate human psychological aspects to enhance the above interaction by observing how users proceed in recommending a service or an item to another person in conventional life. This helps model the human psychology and create accurate and efficient strategies for the recommender so that the recommendations are more on point and personalized for the user.

1.4 Motivation

The main motivation behind this dissertation is the lack of personalization in current recommender systems. It is a general concern that was more easily applicable by developing a movie recommender system, as there are many databases and metadata on movies. Two papers were really inspirational for this Dissertation, [**3**] and [**4**].

1.5 The Problem

Even though personalization in recommender systems is something that has been studied and researched a lot, the study of personalization based on user's actual personality is something new and unique. We believe that a recommender system must know the basic personality traits of the active user and that this will help in the final recommendations. Imagine in real life, having one of your close friends suggesting you a movie to watch. You might like similar things, and you may respect his taste in movies but if he is a calm and shy person in contrast with you being a really active and "crazy" person, then maybe you wouldn't listen to his suggestion and prefer a recommendation from a person that has a similar personality to you.

1.6 Hypothesis

We believe that in the future, every operating system must have a recommender system with a personality test incorporated, that learns the user's taste and reactions and helps both with the operation of the computer and the Internet navigation and the provision of services. We also believe that the user must trust the recommender and it will be easier to do so if he believes that the recommender "knows" his personality.

1.7 The Aim

The aim is first to find an accurate and easy test to examine the user's personality. Next, we aim to find a connection between the personality and the different genres of movies. Then we build a normal Knn Movie recommender system and filter its results based upon the genre preferences the user has (that were exported from the personality test). We then suggest three different sets of recommended movies, one with only the Knn suggestions, one with 50% Knn and 50% personality and one with 80% personality and 20% Knn. The aim is to prove that the 50 % knn and 50 % personality is the one that most people prefer as it combines the best of both worlds, recommendations having the K nearest neighbors in mind and filtering those recommendations with the user's specific personality (genre preferences). This system is the 50/50 movie recommender system.

1.8 Dissertation Outline

This Dissertation is organized as follows:

1. In the first chapter, we provide a literature review of all the different published papers by accredited scholars and researchers. We select the most significant and relevant to our subject, mention their strong or weak points, summarize their main features in our own wording and explain what we improve and where our work fit it in.
2. In the second chapter, we have the analysis of the most important aspects of our dissertation. We explain some important core concepts and introduce the reader to the recommender systems. We provide a brief explanation of the different types of recommender systems. Furthermore, we provide an explanation of the Big Five Personality Test and the human factor in general.
3. In the third chapter, we provide specific details of the design of the 50/50 Movie Recommender System. We provide the basic flow charts and briefly explain how our recommender works.
4. In the fourth chapter, we present the implementation of our system in Python and the programming techniques used. We provide also a small study of the dataset and the interesting information we extract from it.
5. In the fifth chapter, we provide the results of our Evaluation and final conclusions and future work to improve the system.

2 LITERATURE REVIEW

2.1 The Cold-Start Problem

Collaborative filtering is one of the best approaches to develop a recommender system and is the basis of our recommendation engine. As we know, it's a method that offers items to a user, based on items previously rated by other similar users. A big problem though is how to make recommendations for a new user, based upon the fact that he doesn't have any items rated and he can't be clustered with any of the other users. This problem is called the cold start problem.

There are actually many different cold start problems in recommender systems. The 3 basic types are: dealing with a new user, a new item or even a new recommender system from scratch. The one that we study is dealing with a new user.

So, the actual problem is that when a new user registers, we simply don't have a profile, meaning we don't know his preferences. That means that the recommender cannot operate as it doesn't know what to suggest to this user. Now, if the system is not personalized, this makes everything easier because now the recommendation engine can just present items that are based on what is cheapest, most popular, most viewed and generally, whatever the user prefers currently. For example if you visit a mobile phone e-shop, you can simply ask to see the top most sold phones currently and the recommendation engine will provide this for you. No personalization is required for the new user.

Now, to personalize the user, there are many things to consider and many ways it can be done. First and most simple way is to suggest the most popular items the user is likely to be interested in. If for example, in a movie recommender system, a user specifies that action movies are his favorite genre, then we provide him with the most popular action movies. As the user reacts to these recommendations, the engine gathers the ratings he provides and helps personalize the user and make unique recommendations later on. This implies that the system learns with the more ratings the user provides but the danger of this method is that it turns out to be too biased and domain dependent because for example , the recommendation engine might suggest only action movies, something that might be confusing and boring for the user.

Another way is to provide recommendations based on information gathered from other websites, or even from the recommender, like country of origin, age, sex etc. This will make the clustering much easier and accurate as for example young people tend to watch more comedy movies than dramas, so a user that provides all this information is more likely to receive a better recommendation. But again this kind of information is so open to interpretation that it still is confusing to make accurate recommendations. In this dissertation we went a step further and included a personality test so that the recommendations are even more personalized.

Last but not least, one of the most modern methods to treat the cold start problem is to integrate information derived from social networks like Facebook, Google Plus etc. That way you can build a basis for personalization but many people find this method kind of dangerous because you need to provide personal information for it to work, for example you have to provide access to your Facebook profile. As we, see later in this chapter, this method is not that accurate because the recommendations are based on the friends you have on social media and what they like. This is not an easy task as we all know that people tend to add many different people on social networks, even people that they never met or even know. So that for sure can lead to an inaccurate recommendation.

All of these methods have two main goals. First, is to get you to use the recommender. The other is to push you into some sort of activities like answering questions or picking favorite items so that you will provide explicit or implicit ratings and accurate feedback. That information will be later user to personalize even further the recommendations that the active user gets. The main goal of this dissertation is to be able to provide recommendations both by rating some of your favorite movies and by using the Big Five personality test because each personality has a different preference and tends to like different genres of movies.

So, there are many solutions that have been studied in the past and in this part of the dissertation we will review some papers that helped solving the cold start problem in our recommender.

2.2 Literature On The Cold-Start Problem

2.2.1 Using social networks to improve movie rating predictions

In [5], we can see an attempt to start incorporating social media as a way to predict the rating of a movie. It suggests the use of user similarity and not the usual item-based approach to predict the item preferences of the user. The dataset used is a collection of different user ratings and social networks that were provided by Flixster.

In this approach we have a set of users and movies and a set of training examples. Then a matrix is created to represent the Social Media and the interaction between the users. If two users are friends on Facebook for example, then the matrix's appropriate field is 1 and if they are not friends its 0.

The statement this paper makes is that if two users are friends on Social Media, then they have similar tastes. This is the approach the author uses to treat the Cold Start Problem. Specifically, the author states "The premise here is that users who are friends with each other will tend to rate similar movies similarly". This is an overrated approach and we can see that also in the conclusion of the paper.

The author uses the K-nearest algorithm with the addition of the Social Media Friendship similarity. On the following formula, s is the Social Media Friendship similarity, sim(u,v) is the similarity measure between two different users u and v and is on the range from 0 to 1, w is 1 if u and v are friends and 0 if they are not, and sim(u,v). This formula is always close to 1 if the 2 users are friend on Social Media.

$$s = sim(u,v) + \left(1 - sim(u,v)\right) \text{x} w$$

Equation 2.1: Social Media Friendship Similarity

Now, the author treats the Cold Start Problem with the Social Media Similarity in mind instead of using the average rating for a movie when a new user enters the system. The paper states that the best method is to "use the user's friends to gather a set of similar users from which to take an average rating".

As a conclusion, this paper doesn't produce the desirable results. As we know, the effectiveness of an algorithm is based on the value of the Root Mean of Squared Error and in this paper the Social Media Friendship Similarity produce a RSME of 1.48 which is worse than simply using the k-Nearest Neighbor algorithm

In this Dissertation, one of the first ideas of dealing with the Cold Start Problem was to use Social Media as the previous paper. We believe that asking the user to rate at least 20 of his favourite movies and using a personality test will be more successful.

2.2.2 Recommendation of TV shows and movies based on Facebook data

In [**6**], the authors use specifically Facebook data to help solve the Cold Start Problem in recommending movies and TV shows to new users.

The main point of this paper is that eventhough a recommender might recommend popular movies or tv shows to a new user, this is not personalized at all because of the fact that if a movie for example is overhyped it doesn't mean necesseraly that its aprropriate for the specific user. The authors try to solve this problem by looking at a person "likes" on Facebook and assume that there is a correlation between the different tastes in books, music etc and the users taste in movies or TV shows. They try two different methods, one using unsupervised learning and one using supervised learning.

In the first method a K-means clustering algorithm is used to cluster the different Facebook profiles and then recommend specific movies and TV shows to every

different cluster. This simulates real life friend relationship circles because people in the same cluster means that are people which are more similar to each other and probably sharing similar interests. After clustering, we can see which movies and TV shows are famous in each cluster and 10 out of those are recommended to the users.

As usual, the number of clusters was a huge factor, especially when you have a huge dataset so different numbers of clusters varied from 3 to 40 were tested. Big factor was also the Use of Latent Semantic Analysis to reduce the size of the matrix in half.

The authors define the accuracy for each user as the division of hits with the number of shows per movies they recommended multiplied by 100.

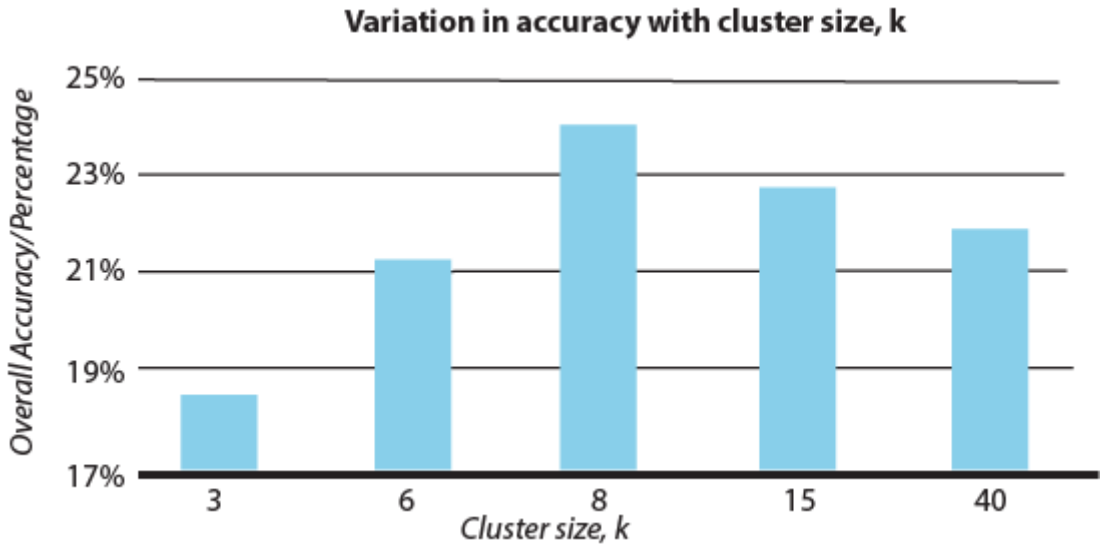
Accuracy for each user=(hits÷number of shows/movies we recommended)*100

Where : **Hits=total number of recommended shows/movies that were listed in their profile**

Number of shows/movie we recommended=10

Overall accuracy=accuracy for each test user ÷number of test users

After many experimentations with the number of clusters they chose k=8 as it was the one with the most accuracy (24%). You can see the results on the following picture:



Picture 2.1: K number

A really big factor in the second method is the data processing. For every different user, the authors collected all their "likes" apart from movies and TV shows and combined them into a long string. This string is called the document of the user, which is then tokenized. They then gather all tokens from all the different documents and form a vocabulary. Next step is the creation of a document-token matrix consisting of the documents, each of which corresponds to a different user, and the tokens of the vocabulary. For example, if the (i,j)th element of the matrix is equal to 1 that means that the jth token of the vocabulary exists also in the ith user's document. Last, they apply the Inverse Document Frequency weighting so that the least important tokens are the ones that appear more often.

Then, they first use a supervised algorithm to recommend the top genres for every user and then recommend specific movies and TV shows out of those. Each user is labeled with the genres they prefer. Important is the use of a weight which helps prioritize the options of the user. According to the paper, if a person liked 2 shows, for example, his first best option is a drama and comedy show and his second best option is a comedy show, then for this user drama genre has a weight of 1 and comedy genre a weight of 2. The bigger the weight, the most likely it is to like a genre. Last, a person (document)-genre matrix is created so that they can get the probability that a term in a vocabulary of a user is directly associated with a genre.

The algorithm they pick to use then is the Naïve Bayes algorithm which predicts the probabilities that each user might like one genre based on the words in their document. Then, they recommend the top five genres and specifically the two most popular movies or shows for those specific genres to all the users.

The results of the second method are not so satisfying. They get 70.98 % accuracy on recommending the top 5 genres but they get a 32.84 % on the two movies/TV shows from these top 5 genres. A good idea would be to further use machine learning to specify more personalized movies for each user and not just the 2 most famous movies on each genre. This will demand the use of content based algorithms for each user and it's something that will be used in this dissertation.

Like the previous paper, the authors mention that their basic problem is the sparse database of Facebook or generally Social media which is one of the main reasons why we don't prefer using those datasets to solve the cold start problem. Specifically in this paper, a big problem was also the processing of the text during the second method, where many users might have misspelled even the simplest word, making it really hard to build the appropriate documents.

2.2.3 Cold-start Problem in collaborative recommender systems: Efficient methods based on Ask-to-rate technique

In [**7**], the authors provide a review on how to use the ask to rate technique to eliminate the cold start problem plus some different implementations of the technique. Generally, these methods are categorized in adaptive and non adaptive.

They provide a method to use the K-nearest neighbour algorithm which was an inspiration for the basic recommender of this dissertation. So to implement the K-nearest neighbour on our recommender, the Pearson Correlation is used first to find the similarity between the active user and all the other users. The Equation for the Pearson Correlation can be seen below.

$$sim(u_t, u_i) = \frac{\sum_{m=1}^{h} (r_{u_t,a_m} - \overline{r_{u_t}}).(r_{u_i,a_m} - \overline{r_{u_i}})}{\sqrt{\sum_{m=1}^{h} (r_{u_t,a_m} - \overline{r_{u_t}})^2} . \sqrt{\sum_{m=1}^{h} (r_{u_i,a_m} - \overline{r_{u_i}})^2}}$$

Equation 2.2: Pearson Correlation 1

Then, an aggregation function is used, to find the predicted rating for an item based on the ratings that the K nearest neighbours have provided. In our dissertation we use this function to predict the ratings that the active user will provide to all the movies of the database. The aggregation function equation can be seen below.

$$prediction(u_t, a_t) = \frac{\sum_{h=1}^{k} (r_{u_h, a_t} - \overline{r_{u_h}}).sim(u_t, u_h)}{\sum_{h=1}^{k} |sim(u_t, u_h)|} + \overline{r_{u_t}}$$

Equation 2.3:Aggregation Function

This paper was also really helpful to make a decision on what movies to present to the new user. In most movie recommendation systems, by the time a user registers, the system demands from the user to rate some movies so that the recommendation engine will start to work. The system must be really cautious about those first movies that it presents, because they need to provide useful information about the user, before the user actually uses the system.

The paper suggests that the best method to make a profile of a new user is to ask for information right away. This is done by presenting items, in our case movies, and ask the user to rate them. Then the collaborative filtering system treats the new user normally and predicts the ratings of the movies he hasn't seen.

The authors test different techniques based mostly on user effort and Recommendation accuracy. The following table shows the different methods and their evaluation in online and offline experiments. We can generally see that IGCN, Entropy0 and Popularity are pretty high rated.

Table 2.1: Method Scoring

Methods	User Effort	Recommendation Accuracy
IGCN	★ ★ ★ ★	★ ★ ★ ★ ★
(Log pop)xEnt	★ ★ ★	★ ★ ★ ★ ★
Entropy0	★ ★ ★ ★ ★	★ ★ ★ ★
HELF	★ ★ ★	★ ★ ★ ★
Popularity	★ ★ ★ ★ ★	★ ★ ★
Item-Item	★ ★ ★ ★★	★ ★
Entropy	★	★ ★
Random	★	★ ★

Two of the above methods were taken into account for our recommender, popularity and entropy. The Popularity strategy is simply taking into account the movie's popularity by counting the number of overall ratings this movie has. We take into account all users and all kind of ratings from 1 to 5. This is a really easy method to implement and its really good for the user as it minimizes his effort more than any other method, but the big problem is that it may create biased opinions on the database. This happens because the more ratings a movie receives, meaning that its really popular, the more likely it will be presented to the user, eventhough it might not be the best choice. This method favours famous movies and keeps back unfamous movies that the user might like, creating unequal distribution of ratings in our database.

Pure entropy is another method that we concidered it might be a good idea to try on our recommender eventhough it doesn't score well in this paper. With Pure entropy, the system suggests the movie that provides the most information for the recommender, eventhough that may mean that the movie is one totally unknown to most people.

In the future, we will try to use the balanced strategy which is the popularity score multiplied by the entropy. This method is the combination of Pure Entropy and Popularity methods and although it was easy to implement, we didn't have the appropriate time to try it.

Another really interesting method that we will implement in the future and was inspired by this paper is the Entropy0 which is the Pure entropy method but with taking into consideration the missing values. This simply means that non-ratings are now filled with 0 whereas 1-5 is the usual rating scale. He follows the formula provided in this paper :

$$Entropy0(a_t) = -\frac{1}{\sum_i w_i} \sum_{i=0}^{5} p_i w_i \log(p_i)$$

Equation 2.4: Entropy

Where w=0.5 is the weight of identifying missing values and Wi=1 in the range of 1-5 because this provided the best results. If we have Wi=0 then the Entropy0 turns into Pure entropy of course. This method is meant to be slightly more successful that the Popularity or Pure Entropy method.

Generally, there are two methods to ask your user information, adaptive and non adaptive. Non-adaptive is when the same information or questions are provided to any new user. This is what MovieLens does, using the popularity stragey, where you suggest a movie based on how many users have rated it. This is what we also tried on this dissertation but as we said before, we found out that although the implementation is easy, the results are biased as people tend to like and rate high, movies that are popular and not neceserally good.

The adaptive method is when the questions for every new user are based on his history or even, in our case, psychological profile. We tried the item-item personalized method where movies are proposed until the user rates at least one. We actually demanded from the user to rate at least 20 movies as this is a basic step for the recommender to work. If the user rate 20 movies with various ratings, from low to high, then this method is problematic with the accuracy as it does not identify movies that the user actually likes. This is why we ask the user to give us his 20 favourite movies, because that way we have a clear idea about what the user likes and recommend him movies based on his highest rated movies .

We also thought about presenting first to the user the personality test, and then suggesting him movies based on his top favourite genres, but again this might lead to biased results. The personality test should be an extra help for the Knn recommendation engine and not dictate the profile of the user so much .

What we finally did in our recommender is to find the most rated movies in the Movielens dataset and present this movies and total random movies alternately. You can see that on 5.7.

2.2.4 Getting to Know You: Learning New User Preferences in Recommender Systems

In [**8**], the authors study six techniques that can be used in collaborative filtering recommender systems to solve the cold start problem. They both try offline and online experiments and they come to the conclusion that each technique affects the user in a different way, both in the user effort and accuracy of the predictions.

They stress the importance that the recommender system should ask the user to rate items that he is more likely to have an opinion about. Minimizing the effort of the user is a critical part , but at the same time, good recommendations must be provided.

One approach they suggest to the cold start problem is that pre-made user categories can be created and the new user can be assigned to one of them. This part was the inspiration to use the Big Five personality type test at, as it helps identify in which user category the new user belongs to. Next step is to combine this with some actual ratings of movies, something that the authors didn't concider. They specifically quote that "When these models are accurate they can be quite useful, but the premise of personalized recommender systems and collaborative filtering is that a person's preferences are a better predictor of other preferences than other attributes". In our case this is not an issue because we know the personality type of the new user, and we suggest him specific movies to rate that this kind of personality usually likes.

Again, the item-item personalized method is stressed in this paper, stating that it's probably the best method to solve the cold start problem. Although it outperformed every other method, it didn't provide the best recommendations but it was the best considering the user's effort. What they suggest is that item-item personalized method is "too much personal", presenting movies that are too obvious that the user will like. For example, if a user rated a movie highly, then it will be recommended to him all the sequels of that movie. Again, we treated this problem by providing both famous and random movies for the user to rate in 5.7

In this paper the authors also imply that users in their research prefer using techniques that allowes them to rate many movies per page and not many pages with less movies between ratings. This is the reason we display a 20 movie list to the user and give him the opportunity to reload this page anytime he wants.

Last but not least, they mention that eventhough popularity and item item strategies are really effective and user friendly, there definitely needs to be an amount of randomness to the way the movies are presented to the new user. Randomness should be treated with caution though because it may lead to excessive user effort and making the recommender really difficult for the user.

2.2.5 Learning preferences of new users in recommender Systems: An information theoretic approach

In [**9**], the authors provide some information theoretic strategies to minimize the new user problem. They state that the system must make sure that the user does not log off the system because he faces a lengthy signup process and that he shouldn't lose trust in the recommender due to low quality recommendations. They both test their strategies online and offline and finally the run an online experiment with actual users on a live recommender system.

They state some pretty interesting observations about the way these techniques are classified and they group them in either human, system or mixed controlled techniques. So, in the context of the recommender system a human controlled technique is when the user selects the items to rate, either by typing the titles or searching over the database. A system controlled technique on the other hand is when the system itself decides on what items will first be presented to the user to rate. And, last but not least, mixed controlled technique is when both human and system controlled techniques are combined into one.

All of the above techniques have their advantages and disadvantages. The user controlled scheme will definitely require more user effort but it will also make the user feel good about the recommender and trust it more. Also, the user might provide biased ratings, meaning that the user might not be actually aware of his exact preferences and pick mostly movies that he remembers and not the ones that he actually likes the most. An accurate system controlled scheme may not put the user in much effort but may sometimes fall short in its recommendations as it doesn't take into account the user's mood and actual preferences. It seems that the best solution is the mixed controlled scheme and this is what we used in our system. We specifically find the top rated movies and let the user pick which one's he prefers to rate. We also provide some random movies so that the picks are less biased and the user's profile will be more universal.

We saw again in previous papers, that user effort and recommender accuracy are the two base factors for the systems success. In this paper, the authors extend the work of [**8**] and study the application of different item selection measures based on information theory for the cold start problem. They test all the different measures and find different set of items which the provide to the new user. Then they record the user's input and see how effective this set of items is to solve the cold start problem. This will be the base for their mixed method system.

They then proceed with the components of their experimental platform and they suggest using both user- based and Item-based K-nearest neighbours algorithm. In our dissertation we only use the user based Knn as we believe that the Item based Knn doesn't offer much to the recommender. They also use MovieLens2 as their experimental platfom where the user goes from page to page where each one contains 10-15 movies to rate. But what they find next on their online experiments is that eventhough the demand for a lot of ratings from the user is requested, the users don't find that to be much of an effort and understand that this way the recommender will operate better. Actually 80% of the users finished the sign up process which is a pretty good percent concidering the effort that is demanded from the user. One user specifically wrote "I understand why it is needed. I know it will make your recommendations more accurate." This part of the paper was an inspiration to ask for the user to rate at least 20 movies and not 10 as we were planning at first.

In the conclusion they suggest working on the future on updating the user profiles based on their age of evaluations, meaning that old ratings must be eliminated or new ratings must have more weight. This is an interesting thought but its something that might again be biased and this is why we didn't experiment with it on our dissertation.

2.2.6 A hybrid recommender system based on user-recommender interaction

In [**10**], the authors create a hybrid recommender system based on the MovieLens dataset combining random and k-nearest neighbor algorithms. They state that the most current recommender systems seldom imply specific user-recommender interaction scenarios in real-world environments and they try to solve this problem with the creation of a new hybrid recommender system. This system will use the random algorithm to solve the cold-start problem, something that was an inspiration in our dissertation but we believe that using only the random algorithm will make the user effort much higher. This is why we suggest to the new user a few random and a few really famous movies to rate at the beginning.

Then they define the recommender behavior in 3 different steps. First, the recommender accepts the user's request, second the recommender presents N recommendations to the user and third the system records the user's choice for further usage. This is also a small preview of how are our recommender operates. They also define the user behavior in three steps. First the user gets the recommendations, second the user checks if the recommendations match his preferences and third the user proceeds with a choice. The user behavior in our movie recommender system is a little different, because the user get a set of recommendations that includes different movies so he doesn't have exactly the choice to see if that set match his preferences. He can just pick a movie from the set he was suggested.

The authors also provide us with assumptions to simplify the whole procedure. This was an inspiration to provide our own assumptions in 5.4.

In this paper, the authors divide the users of the Recommender System into two different categories called the browse and the rate user. Browse user is the one who specifies only which items are browsed and on the other hand, rate user is the one that specifies the ratings to the items. In their study they only consider browse users, which is something that is not helpful for our project, as our recommender does not work with browsing in mind ,but with the users providing actual ratings. You can see below the user and recommender behavior in their system.

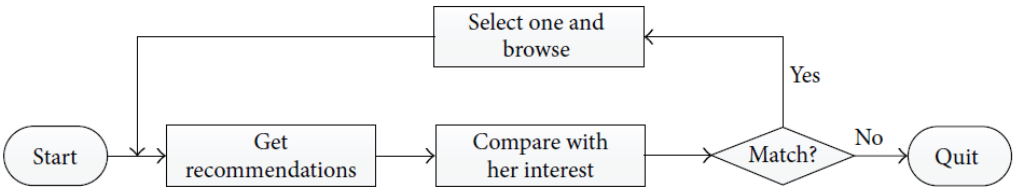
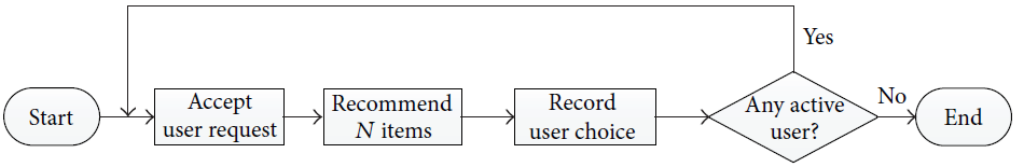


FIGURE 1: The user behavior.



Picture 2.2:The recommender Behavior

Generally this paper didn't contribute much to our recommender system because it is mostly based on the browsing experience of a user using a Movie recommendation system. But, it introduced us to the hybrid approach of using the random and knn algorithms which is something that we used in our own ways to solve the cold start problem. Also, it inspired us to include a chapter that includes some important assumptions about our recommender system.

2.3 Personalization In Recommender Systems

Nowadays, with the Internet being the main source of entertainment, products and services, personalization is a huge factor and a great area to study. Scientists start to incorporate elements of the human psychology in human-computer interaction, and this gets more and more important because computers should work with the users preferences in mind. Some of those elements would be demographic information such as age and gender, personal preferences such as music or movie taste and even psychological aspects, meaning emotions or personality traits.

Computer scientists make a great effort to model the human psychological aspects and include them in recommender systems, so to create better suggestions for the users and make the recommender more personalized. There are many different ways to extract the personality of a user, from his social media activities, or making him answer a simple questionnaire. One might say that this can be confusing or demand a lot of effort from the user, but the user must understand that by giving more input to the system, the system will provide better output, meaning better recommendations. A big problem of course is the low amount of data around people's personality traits, but we hope that this will change in the future.

In this part of the dissertation, we study different ways that were researched in the past to gather and use the user's personality into a recommendation system. Although these papers are really interesting, their lack of data and implementation is something that lead us to make a movie recommender system in Python with some actual Data and provide substantial results.

2.4 Literature On Personalization

2.4.1 Multi criteria pseudo rating and multidimensional user profile for movie recommender system

Paper [**11**] is a simple introduction on how a certain aspect of human factor can be applied on recommender systems. The main subject of this paper is reducing the Sparsity Rating problem by incorporating contextual information such as where, how, with whom and at what time the movie has been seen.

The Sparsity Rating Problem is caused when users in recommender systems rate only a small number of items, and the system cannot calculate and find the neighbors of the active user due to lack of co-rated items. This is why in this dissertation we ask for the user to rate at least 20 movies at the beginning. Then pseudo ratings are being generated and the recommender suggests to the user the one's with the highest value.

In this paper, the authors attempt to enhance the quality of these pseudo ratings by including contextual information of the user. Specifically, they do this on a movie recommender system by applying Naïve Bayes algorithm and multi regression to analyze the contextual information.

They also mention the Without Contextual Information Problem, where the recommender system suggests different movies based on contextual information. It's totally different for the recommender, if someone wants to watch a movie with a friend or alone. If he wants to watch a movie alone, then the recommender will see his ratings, find his neighbors and suggest some movies, but if he wants to watch a movie with a friend, then the ratings and pseudo ratings of this friend will chance the outcome of the recommender, so that it will suggest a movie that they will both like.

Apart from the companion, this paper also presents the place, time and day dimensions, which are pretty interesting and something that we will try in the future. Next step is turning the movie data into separate vectors, which was an inspiration to do the same into our recommender but instead we used the personality dimension.

Moreover, it was really helpful to see that the ratings are then normalized. This is something that we also did in our recommender. The users rate a movie with -1 if they don’t like it, 0 if it's neutral and 1 if they like it. Although this is easy for the user to understand, it needs to be “translated," meaning normalized, into something that the recommender will understand. For example, later it will be really difficult to make equations if a user rated a movie with a 0 because all equations will turn out 0. So, the they convert the values to 0, 0.5 and 1. That way, equations will return 0 only if a user rated a movie as -1 which is actually what we want. In our recommender we followed a similar procedure which you can read in pages 111-112 of the Script.

The use of Naïve Bayes is a little questionable in this paper as this algorithm cannot learn and understand interactions between variables and features. For example, if the user loves to watch horror films and usually enjoys movies at night, the algorithm cannot understand that the user might not like to watch horror films at night.

2.4.2 Making recommendations better: An analytic model for Human-Recommender interaction

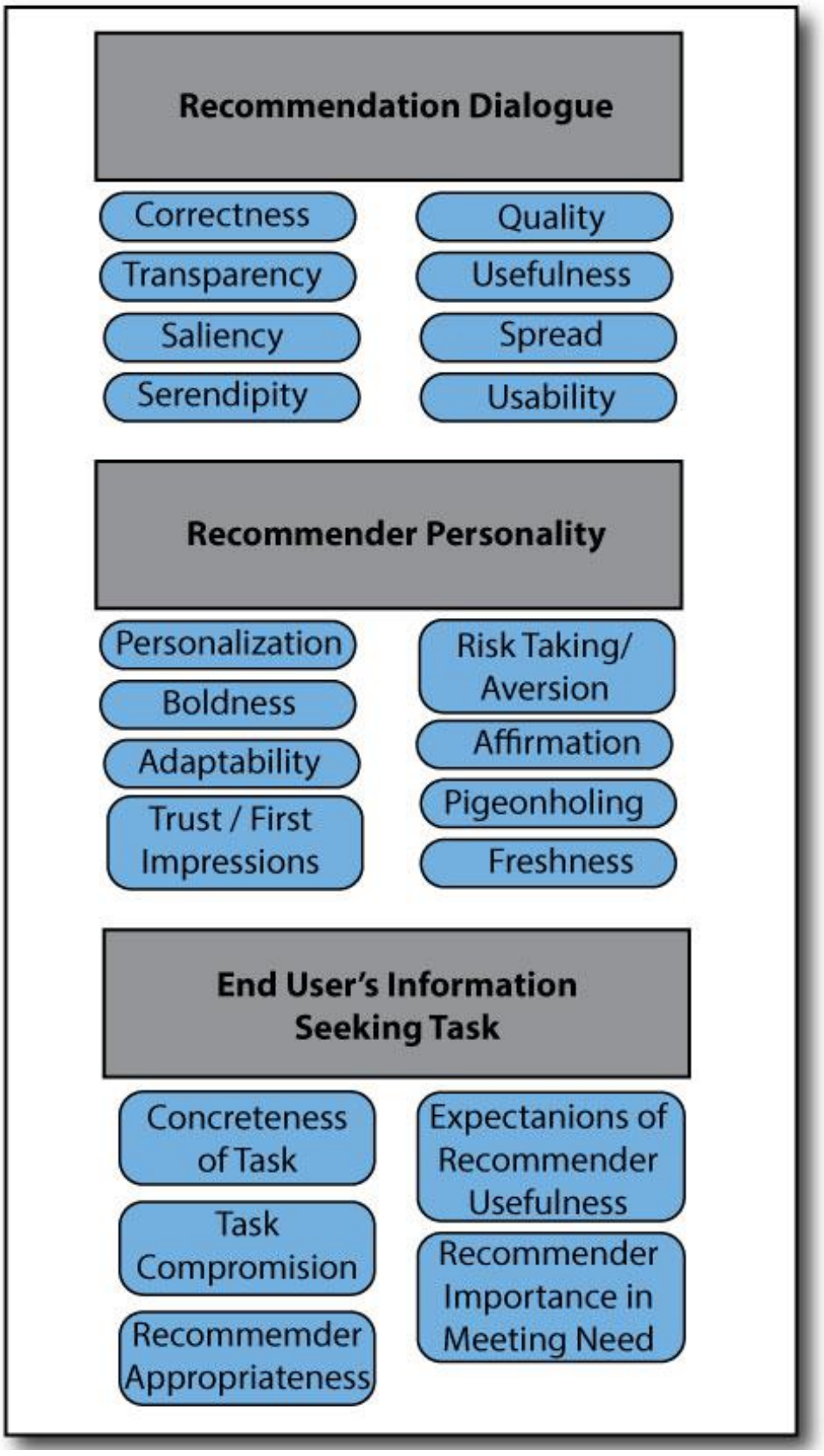
Paper [**3**] is the main reason we decided to study how the Human Factor and personality affects the Recommender systems. As described in the paper, Recommender systems are not always capable of generating good recommendations for the user based only on raw data. To improve the system, the recommender needs to understand the needs and information seeking tasks of the user. The authors provide their own framework, which is called Human-Recommender Interaction (HRI). It's their way to connect the user needs with the recommender algorithms.

Their two main assumptions are first, that recommenders cannot understand the reason a user asks for recommendations and second, that recommenders should be trained to have personalities and interact with the users in a conversational way. The second assumption is the main reason we use the Big Five personality test so that the recommender "knows" and operates accordingly to the user's personality type.

Then, the authors proceed to the idea that there are different types of recommendations based on what the user needs and not based on the casual thought that best recommendation equals to more liked recommendations. They use HRI to understand what the user actually needs, which is different most of the time because people tend to have different needs based on what they search for , when they search for it and why they search for it etc. For example, as stated in the paper, if a student is writing one of his first papers and using a recommender for his research and still feels that something is missing, he is actually looking for validation of his research and suggestion on whether he missed something. The human-Recommender Interaction system makes the assumption that this student is concerned about the effectiveness of the recommender because he is a novice user, so the recommender must adapt and suggest more appropriate recommendations based on this extra information the system generated from the user.

The authors mention that there are three Pillars to HRI shown in the following table.

Table 2.2: Pillars to HRI



First is the Recommendation Dialogue, which is the procedure where the user passes information to the recommender and the recommender returns his recommendations. Second is the Recommender Personality, which is really important and is the perception the user has for the recommender over the period of time. It's actually how the recommender adapts to the user's needs. Last, the End User's Information Seeking Task is the reason the user needs and uses the recommender system. All this pillars are really interesting and might be used in the future to make our recommender even more advanced.

Generally, this paper appreciates the use of recommender systems and suggests that people should establish a relationship of trust with the recommendation engine, but also the engine must be able to adapt to the user needs. There is not any actual implementation as we believe it's pretty difficult due to the fact that they are no such data that will allow testing this model. Some of the characteristics of each of the 3 pillars of the HRI are also represented on the Big Five personality test so although the idea of this paper is pretty good, the lack of implementation and testing makes it a little ambitious. In the future, apart from the personality test, it will also be a good idea to implement a test to examine exactly what the user specifically needs from the recommender. That way the recommender will be more user-centric and we will be able to better analyze and understand the user's needs. With us implementing the Big Five personality test, we managed to emulate the two Pillars of this HRI engine, the Recommendation Dialogue and Recommendation personality in a simpler and more straight forward way, based on the current data that is out on the Internet.

2.4.3 Personality, movie preferences, and recommendations

This paper is a relative small but interesting paper. In [**12**], the authors research the connection between media preferences and personality plus the correlation between the users opinion of a recommender system and the media ratings and viewing histories. They provide a survey out of 73 Netflix users, which is not really that representative, but they manage to show different correlations between different types of personalities and what preferences these personalities have for specific genres of movies

Their result is that Conscientious, a personality trait that is included in the Big Five personality test, is strongly correlated with people who trust and find more use in recommendation systems. Their main focus on this paper is to answer whether there is a relationship between human personality traits and movies preferences and whether these personality traits help the user to gain trust over the recommender

They use the Big Five Personality test and then they collect ratings on different movies recommended by Netflix. This is really similar to our approach in this Dissertation.

They then ask users to answer some simple questions so that they can figure out the user's actual attitude toward the recommender system. By using Pearson correlation between the users personality scores on each of the Big Five Attributes and their answers to the above specific questions, they gather some useful results on the table below.

Table 2.3: Results

	Extra.	**Agree.**	**Consc.**	**Neuro.**	**Open.**
How often do you look over some of the movies that Netflix recommends?	0,06	0,03	**0,27**	-0,14	0,10
How often do you add recommended movies to your queue?	0,19	0,00	**0,24**	0,04	0,00
What percentage of movies in your queue were recommended by Netflix?	0,11	0,10	**0,25**	-0,08	-0,16
How helpful do you think the recommender system is?	0,15	0,15	**0,32**	-0,07	-0,20
How much do you trust the recommender system?	-0,06	0,13	**0,24**	-0,13	**-0,24**

We can clearly see that Conscientiousness is strongly correlated with users who are more positive towards the use of the recommender system. This translates to the fact that Conscientious people rate more movies than the average user, meaning that they trust the recommender more, so the recommender has more data to analyze and make recommendations for those users.

People with Conscientiousness are people who like to plan, think very carefully and are well organized. The authors believe that there is a positive correlation between this kind of people and recommender systems because recommender systems will help organize their viewing experience. Recommender systems organize a plan, with movies to watch and this people trust that plan and as a result they take more suggestions from the system.

A negative aspect the authors found about this correlation is that Conscientious people respond very badly when a recommender suggests a movie that they actually don't like. They are very sensitive to big negative changes so that one incident can negatively impact the user's impression that the system can actually help and provide a plan for movies to watch.

The authors didn't manage to find a correlation between Conscientiousness and genre preferences and this was an inspiration to research the Big Five Traits and how they relate to genre preferences. In this dissertation, strong correlations between the user's personalities and movie genres are created, using a pretty big data set, which helps solidify more the results.

Even though the authors manage to find a great correlation between the user's personality and the recommender system, we take this study even further and we study all the Big Five Traits and how they interact with the user's genre preferences. The questionnaire they use after the personality test is done is also a great idea and something that we will improve and include in the future.

2.4.4 Movie recommender system using the user's psychological profile

In [**13**], the authors create a movie recommender using a hybrid recommender system and the users personality, pretty similar to the recommender we created in this Dissertation Project.

Their recommender system is called Movie Recommender and uses both collaborative and content based filtering methods plus the analysis of the user's personality and psychological profile. Their main goal is to prove that there is an actual correlation between personality traits and movie preferences, pretty similar to what we are trying to achieve.

They use the IMDB Top 100 Greatest Movies of All Time as their database and collected the movie title, the actors, its genre and score received on IMDB. In the field of Data Mining and Recommender Systems, a database with 100 elements is not big enough to base an actual conclusion from the operation of a Recommender System, so this is why in our recommender system we used the Movielens dataset which has a lot more data. In the next table you can see the distribution of movie genres on the dataset they used.

Table 2.4:Movies/Genres

Movie Genre	Number of movies having the genre
Adventure	18
Drama	56
Romance	22
War	9
FilmNoir	6
Comedy	18
SciFi	8
Horror	10
Action	13
Animation	2

Then they proceed to compute four features for the hybrid recommender and they do that by storing the history of all the previously rated movies and four different personality scores, for all the users.

A big flaw in this paper is the choice of the questions that are used to come up with the user's personality. The authors pick a personality test with fewer than normal questions but it's not based on an actual research. It's good that the authors try to make the recommender faster and simpler so that the users will actually complete the Personality test but we believe that they are not in the position to understand how the Personality test actually works as there are many psychological factors involved. So, this is the reason why we use the world known Big five personality test in our recommender system. But generally, their personality test uses the four temperaments which are sanguine, choleric, melancholic and phlegmatic.

Also another big problem with this paper is the fact that they make up how to evaluate the test and the answers to figure out what type of personality the user is. This was a big concern in our paper and this is why we used the Big Five Personality test, as it is the only personality test that provides the calculations that needs to be done to find a user's personality , [**14**]. They come up with the following table.

Table 2.5: User Profile and Genre Preferences

Psychological Profile	Characteristics	Movie genres
Choleric	extroverted, hot-tempered, quick thinking, active, practical, strong-willed, self-confident, independent	Horror, War, Action, C omedy, Romance
Sanguine	extroverted, fun-loving, impulsive, entertaining, persuasive, easily amused, optimistic, receptive, animated, excited	Action, Adventure, Comedy, Drama, Romance
Phlegmatic	introverted, calm, unemotional, easygoing, slow, indirect, practical, patient, persistent, consistent	Sci-Fi, Tragedy, Cult, Drama. Film-Noir
Melancholic	introverted, logical, analytical, factual, private, reserved, timid, self-sacrificing, gifted	Animation. Film-Noir, Parody, Cult

Then they use collaborative filtering in a similar way also used in this Dissertation. They first make an algorithm to find similar movies but they don't mention where their similarity is based on. We believe that there is no particular reason to find the similarity between the movies as this will return biased recommendations. For example Rambo 1 and Rambo 2 should be really similar according to their algorithm but it's not applicable to recommend Rambo 2 to a user as this is not an intelligent recommendation, based on the fact that you already asked the user to answer a personality test. It might seem a little disappointing to spend time answering a personality test and then get the most obvious recommendations. They also use the Knn algorithm the way we do.

As we can see on the final formula below, they put twice as much importance on the movie similarity measure. Based on what was mentioned previously, this will make the recommendations more biased and this is why we use three different formulas on our dissertation, one with only the Knn, one with 50% personality and 50% Knn and one with 80 % personality and 20 % Knn. Also, it seems that this recommendation engine suggests only one movie which is pretty limited.

$$movie_{rec} = \underset{m \in movies}{arg\,max} \begin{pmatrix} 2 * sim_{Movies}(m) + sim_{Profile}(m) \\ + sim_{Ratings}(m) + sim_{KNN}(m) \end{pmatrix}$$

Equation 2.5: Movie Recommendation Formula

They state that their basic problem was coping with the cold start problem due to the fact that they have insufficient number of data and users hesitating to spend time filling the personality test. This is something that we managed to solve using the MovieLens database and using a more accurate and trustworthy personality test.

They specifically mention" The fact that our system is based on first evaluating the psychological profile of the users makes it impossible for a reliable evaluation of the system's accuracy. Although there are several free datasets available (such as the ones from MovieLens5, for example), they lack the psychological profile of the users and therefore cannot be used to evaluate the system."

This is not actually a problem and it's an understatement. There is no need to have the psychological profiles of the users in your dataset because the main focus is to find what personality type is the current user and use this as a mean to understand what genres he prefers. So you don't actually need personality types of all the users but you just need to determine the type of the active user so to come to a conclusion about the movies genres he likes. Then, as we do in our recommender, you use that movie genres preference to filter the final recommendations.

Also one might say that if you have the psychological profiles of all the users in your dataset, you can use that to find K nearest neighbors to the current user but this is also not important because we already use Knn for all the users and the current user. That way the recommender is less biased because he gets recommendations from all the different kind of user personalities and then they are filtered based on his genre preferences. This means that he can get a movie that it might not belong in his favorite genres but it will belong in his K-nearest neighbors' favorite movies, so it might be an outsider that he actually likes.

2.4.5 Relating Personality types with user preferences in multiple entertainment domains

This specific paper is the foundation of this dissertation project. It provides useful data that help us achieve the connection between the human personality and the movie recommender. Here follows a summary of what it's included in this paper.

In [**4**], the authors present a study between different user personality types and their preferences towards movies, tv shows, books and music. They analyzed a pretty big number of Facebook user profiles (almost 54.000) that contained the user's personality scores in the Big Five Personality test and their interests in the entertainment domains previously mentioned. This paper is one of the few that provides a through data analysis of different users and comes up with data statistics that can be used into our recommender.

They make a great point on the importance of personality, as it is a combination of characteristics and qualities that are unique to every user and develop an individual way of thinking and behaving. Having this in mind they proceed with the Five Factor Model, which is what we also call the Big Five model, and use this to establish each different user's personality.

The authors based their research on a database released by a tool called myPersonality, a Facebook application where Facebook users take psychometric tests. They restrict their analysis to a subset of this database and after implying some data mining techniques they come up with a dataset that includes the top 16 genres in movies, music, books and tv shows, 53,226 users (where almost 60% are female an 40% are male) and of course the Big Five Personality scores.

They then analyze the dataset and they find important and meaningful correlations between the Big Five personality types and the user's movie preferences. They provide the following table which we used in our own dissertation .

Table 2.6: Correlation between Genre and Big Five Traits

	All users					
MOVIE GENRE	**OPE**	**CON**	**EXT**	**AGR**	**NEU**	**#users**
action	3.87	3.45	3.57	3.58	2.72	2488
adventure	3.91	3.56	3.54	3.68	2.61	179
animation	4.04	3.22	3.26	3.35	3.02	85
cartoon	3.95	3.33	3.49	3.57	2.81	957
comedy	3.88	3.44	3.58	3.60	2.75	3969
cult	4.27	3.10	3.45	3.40	3.16	38
drama	3.99	3.43	3.66	3.60	2.86	905
foreign	4.15	3.46	3.47	3.54	2.81	112
horror	3.90	3.38	3.52	3.47	2.91	2284
independent	4.31	3.59	3.51	3.55	2.69	104
neo-noir	4.34	3.35	3.33	3.37	2.97	92
parody	4.13	3.36	3.35	3.28	2.73	25
romance	3.84	3.48	3.62	3.62	2.85	776
science fiction	3.99	3.55	3.33	3.57	2.73	215
tragedy	4.40	3.34	3.27	3.52	3.11	26
war	3.82	3.51	3.49	3.50	2.71	148
	4.05	**3.41**	**3.46**	**3.51**	**2.84**	

This table shows the personality based Traits (Openness, Contentiousness, Neuroticism, Extraversion and Agreeableness) for the 16 different movie genres. Each row of the table is a vector and the values on each cell are in the range of 1-5 and they represent the average score of the Big Five personality traits of the users who had liked the corresponding genres.

The highest the score of a column, the greener the column will be and the lowest the score of the column, the redder it will be.. A remarkable notice is that some genres were erased either from the movielens dataset or from this table so that we have the same genres throughout the dissertation.

We generally notice that users with high openness have a tendency to like tragedy, neo-noir, independent, cult and foreign movies. High conscientiousness corresponds to users who favor independent, adventure and sci fi movies. High Extraversion also corresponds to users who favors drama, romance, comedy and action movies and high Agreeableness means that the users like Adventure, romance, comedy and drama movies. Last, high value in Neuroticism means that the users prefer cult tragedy and animation movies. We should point out that all the preferences we mention for each Big Five personality trait and each genre is in order of preference and not random and this is what makes it interesting.

Furthermore, the authors apply the Apriori algorithm to derive some association rules from the dataset. This might be a good idea to try in the future. Their results can be seen in the table below.

Table 2.7:Apriori Rules

MOVIES	Rule	Confidence	Support
All users (support ≤ 20.30 %)	$con \in (3, 3.25] \wedge agr \in [2.55, 2.87] \rightarrow$ cult	67 %	1.87 %
	$ope \in (3.6, 3.80] \wedge ext \in (3.35, 3.62] \wedge agr \in (3.52, 3.85] \rightarrow$ comedy	67 %	1.87 %
	$ope \in (3.8, 4] \wedge con \in (3.25, 3.5] \wedge agr \in (3.2, 3.52] \wedge neu \in (2.85, 3.17] \rightarrow$ horror	67 %	1.87 %
	$ope \in (4.88, 5] \rightarrow$ tragedy	63 %	2.50 %
	$ope \in (4.4, 4.6] \wedge ext \in (3.62, 3.89] \rightarrow$ foreign	63 %	2.50 %
	$ope \in (3.6, 3.8] \wedge ext \in (3.62, 3.89] \wedge agr \in (3.2, 3.52] \rightarrow$ horror	57 %	2.19 %
	$ope \in (3.6, 3.8] \wedge ext \in (3.62, 3.89] \wedge agr \in (3.20, 3.52] \wedge neu \in (2.53, 2.85] \rightarrow$ horror	57 %	2.19 %
	$ope \in (3.6, 3.8] \wedge agr \in (3.2, 3.52] \wedge neu \in (2.53, 2.85] \rightarrow$ horror	50 %	2.50 %
	$ope \in (3.8, 4] \wedge ext \in (3.62, 3.89] \wedge agr \in (3.52, 3.85] \rightarrow$ comedy	50 %	2.81 %

Here we can see the rules that have the most confidence. For example, the first two rules imply that if a person has high contentiousness, high Extraversion and high Agreeableness he would like comedy with a confidence of 67% and Support 1.87 %. All this rules are really interesting but we don't think that add extra value to the already findings. What we will do in the future is to apply the apriori algorithm and derive association rules that include the information of the user that is included in the Movielens file and the genre. For example it will be interesting to extract rules that include the gender, age and occupation with the genre preference. That way we can imply those rules on marketing strategies and online ad campaigns.

In the following and last table we can see the similarities between the different types of personalities and the different genres. The color of the cells is the values of the Euclidean distances between the genres associated to the user personality stereotypes. For example a person who likes comedy and action genres has similar Big Five personality based profiles. Also, people who like romance movies have similar personalities to people who like comedy and action.

Table 2.8:Similarities

MOVIES/MOVIES	action	adventure	animation	cartoon	comedy	cult	drama	foreign	horror	independent	neo-noir	parody	romance	science fiction	tragedy	war
action																
adventure																
animation																
cartoon																
comedy																
cult																
drama																
foreign																
horror																
independent																
neo-noir																
parody																
romance																
science fiction																
tragedy																
war																

3 COLLABORATIVE FILTERING RECOMMENDER SYSTEMS

3.1 Concepts And Vocabulary

There are many common concepts and vocabulary used in the various recommendation methods. In collaborative filtering though, many of those concepts are being used to describe the problem and the requirements on the system.

So, in a Recommender system that uses collaborative filtering, we have the users which they provide various ratings for different items. In our movie recommender, the *users* are the people that rated all the various movies in the Movielens Dataset (plus the active user), and the *items* are all the different movies. Those users provide *ratings* for these movies, where rating is a general expression of preference. Those ratings can be either *explicit*, meaning something that the user entered himself or *implicit*, something that was calculated from the user's behavior. In our system, the ratings are in a *scale* from 1 to 5 but generally the ratings of items in a recommender can take many forms. An even simpler way would be to use a like/dislike system, for example, where the user enters 1 if he liked the movie and 0 if he didn't but that would make our recommender less accurate. This is a pretty good method for other kinds of recommender systems like the ones used in e-commerce websites, where you only care if the buyer, for example, bought a specific product or not.

What happens next with the users and the items is that we form a *ratings matrix* like the example shown below. Here we can see three different users and their ratings for different movies. *Nan* is placed when the user hasn't expressed his opinion about the movie, meaning that he simply haven't seen the movie, or he has seen it and didn't manage to rate it yet.

Table 3.1: Rating Matrix example

	Titanic	**Alien**	**Terminator**	**Forrest Gump**
Nick	5	2	Nan	4
John	1	5	5	Nan
Peter	3	Nan	3	Nan

The first important goal of a collaborative filtering recommender system is to make *predictions*, meaning to predict whether a user likes or not a specific item. For example, again, if we check the ratings in the above table, we can predict that user Nick won't probably like the movie Terminator, as he had rated two similar movies (Titanic and Forest Gump) highly and also rated Alien with a low rating. Alien and Terminator are both action/horror movies, so we can see that generally Nick doesn't like action/horror movies and likes casual/ drama movies like Titanic and Forest Gump. As a conclusion Terminator will not be a good option for Nick.

Second and most important goal of a recommender system is to make *recommendations* to the different users. That means, predict first and then present a list with the best items that the user will most probably like. In collaborative filtering, this happens by looking what similar users with the active user like.

3.2 How It Works

The most common way of getting a recommendation in real life is by asking a friend for his opinion, preferably someone who usually likes the same stuff as you do. In this Dissertation, for example, we are looking for "friends" that have a similar taste with the active user in movies.

This is exactly the idea behind collaborative filtering, and that's what the word collaborative actually means. It's a way in which users help each other out in navigating the catalogue of different products, in our case, movies, so to find things that they like.

So the basic premise of Collaborative Filtering is that if two users have the same opinion about a bunch of products, then they are likely to have the same opinion about other products too. Collaborative Filtering is a general term and any algorithm that relies only on user behavior (history, ratings, similar uses, etc) is a Collaborative filtering algorithm.

The objective of the algorithm is to normally predict the active user's ratings for products he hasn't yet rated. The input for the algorithm is usually a database that contains different users and their ratings for different products in the past. As we previously mentioned, these ratings could be either explicit or implicit. Picking this input and the past ratings of the active user on different products, the collaborative filtering algorithm will predict the ratings for the products the user hasn't checked out yet. With these predicted ratings, you can sort the products and recommend the top picks for the active user..

3.3 Basic Steps

Here follow the basic steps of a simple Collaborative Filtering operation.

1. The set of ratings for the active user is identified.
2. The set of other users who are most similar to the active user, according to a similarity function (in our case Pearson Correlation) is also identified.
3. Then, we identify the products that these similar users liked.
4. A prediction is generated, meaning a rating that would be given by the active user for each of these products.
5. A set of top N products is recommended based upon the top highest predicted ratings of the products in the previous step..

3.4 Taste Assumptions

There are two basic assumptions that data scientists made back then about the different users of the recommender system. The first and most important is the fact that if the active user has a high pair similarity correlation with another user, then we assume that apart from their already rated products that they share similar tastes, they also share similar tastes on future products that they both haven't rated. That generally means that we assume that all users have stable tastes and that the strongly correlated users will like related things in the future. This assumption helps the recommender system work well as it's easier to calculate and keep the useful neighbors of the active user.

The second assumption that helped the operation of the recommender in the past was one which implied that if the correlated users agree on one part of the recommender , they will likely agree to more parts too. This is a really helpful assumption, especially in a movie recommender system like our 50/50 system. That means, for example, that if two users tend to rate highly movies whichinclude Vin Diesel, we assume that they like Action films. Or of course, the opposite.

Those assumptions that scientists made back in the days, led us to make our own assumptions on 5.4.

3.5 K-nearest Neighbors Algorithm

Knn is one of the most famous algorithms in modern data science. It is considered a non-parametric algorithm, meaning that it does not make any assumptions about the data distribution. This is a really important and helpful feature in the modern world, as most data don't follow the typical assumptions made. Knn is also what we call a *lazy algorithm*, meaning that there is a minimal training phase in the data, making the algorithm really fast. This means that Knn keeps track of all the data all the time and bases his decisions on that. This makes the testing phase a really costly procedure.

One of the most important assumptions in the Knn algorithm is the fact that it assumes that all data are in a feature space or even better, a metric space. The data are usually vectors, as you can also see in this dissertation. All data are considered different points in this metric space.

Also, really important is the number k. This is the number of "neighbors" that the algorithm will have in mind when it makes its classification. It's a common technique to make this number equal to the square root of the number of data, but the search for the optimal k is a whole different area of study by itself.

3.5.1 Advantages of Knn over other algorithms

One of the main advantages of the Knn algorithm is its simplicity. It's one of the easiest algorithms to implement as you only need to configure and tune k, meaning the number of neighbors that will be used in the prediction. Numerical predictions are easy to interpret, and you can easily see any given time which neighbors are being used for the prediction.

Another advantage is the fact that it makes explaining the recommendations to the users really easy. In item-based recommender systems, for example, the active user is presented with the neighbors' items and their ratings, so he gets a justification on why these items were suggested to him.

Efficiency is also a big factor in Knn since it's a lazy algorithm and requires no costly training phase of the data. Everything can happen in offline mode, and all the neighbors and recommendations can be stored with little memory usage. This can make the recommender scalable to millions of products and users and can even determine which variables are important, improving the recommendations.

Last but not least, the fact that Knn is an "online" technique is a big plus, especially in recommender systems. This actually means that new items and users can be added at any time without retraining the dataset. This makes the Knn one of the most stable algorithms, in contrast with, for example, the support vector machines algorithm where retraining of the data is required.

3.5.2 Weaknesses of Knn over other algorithms

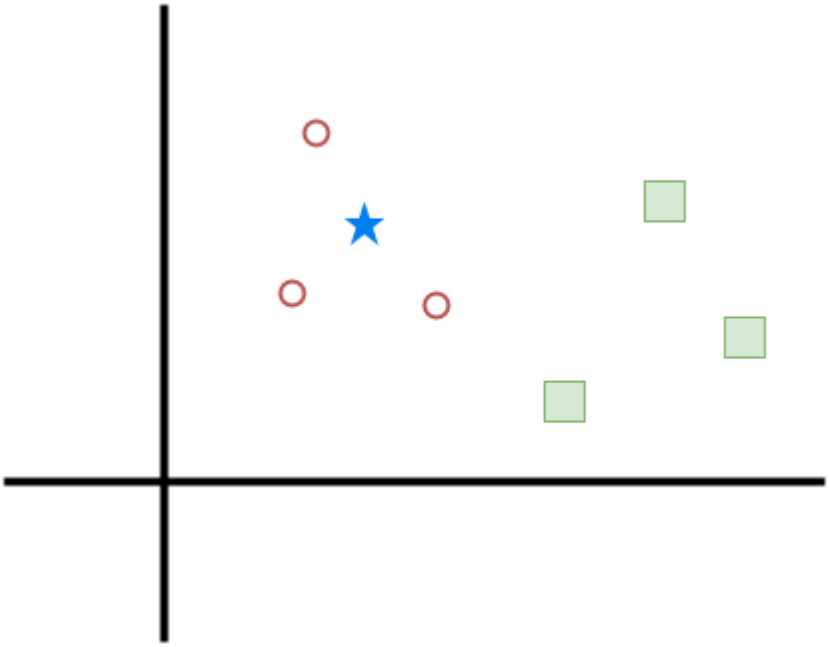
Apart from all the advantages, Knn has also some disadvantages. One major disadvantage is the fact that all the data from the dataset has to be "present" in order to make predictions. This is a time issue as the algorithm has to compare every new item or new user with all the previous and of course, in a dataset with millions of items and users, this process takes a long time. Generally making the predictions is very expensive as you have to calculate the distances between all the different data points.

Furthermore, finding out the correct weights and scaling factors might be a little difficult to determine, as a lot of time, trial and error is needed before finding the right ones.

Last, in Knn we have what we call the "Curse of Dimensionality," which is a problem that occurs when the solution space has more than one dimensions. That leads to big differences between the performance of the algorithm on the unseen data and the training set. In more details, this means that even though Knn works really well with a few input variables, if these inputs increase, then the dimensions also increase, and the more dimensions, the more distance is created between the data points. This is considered an unexpected behavior of distances.

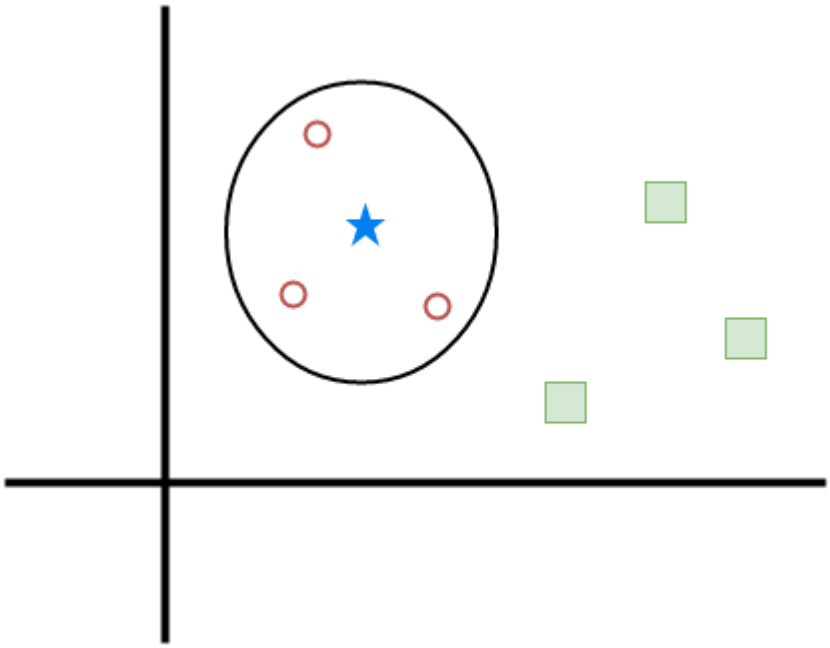
3.5.3 Explanation of Knn operation

Here we will present you with a simple example to understand how the Knn operates. In the following diagram, we have three red circles and three green squares. They represent what we call so far, the data points and also the neighbors of the blue star.



Picture 3.1: Knn Operation 1

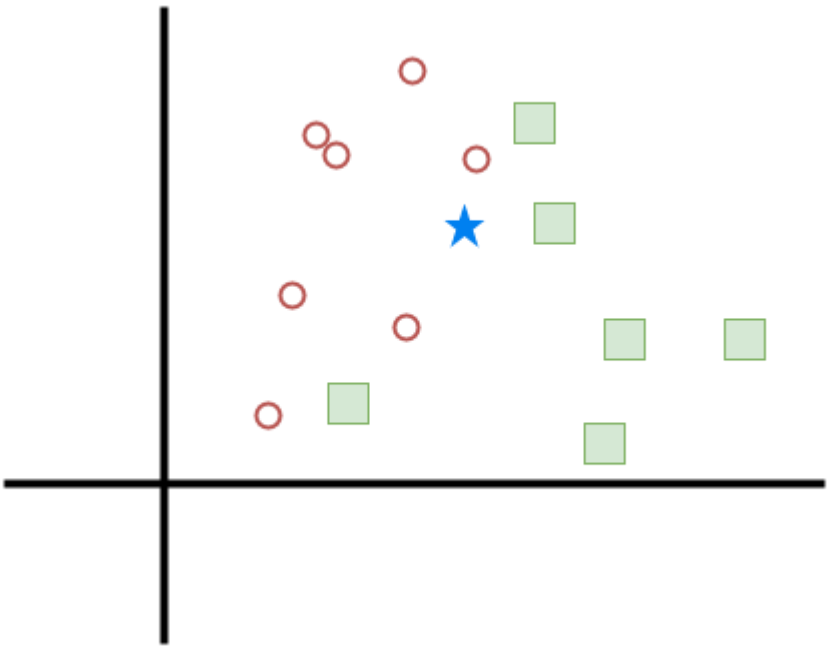
The main point of the Knn algorithm is to find if the blue star is a red circle or a green square. Now as we said before we need to arrange the value of k, meaning how many nearest neighbors we will have in mind when we will try to figure out where the blue star belongs. For now let’s say that k=3 and that leads us to the following picture.



Picture 3.2: Knn Operation 2

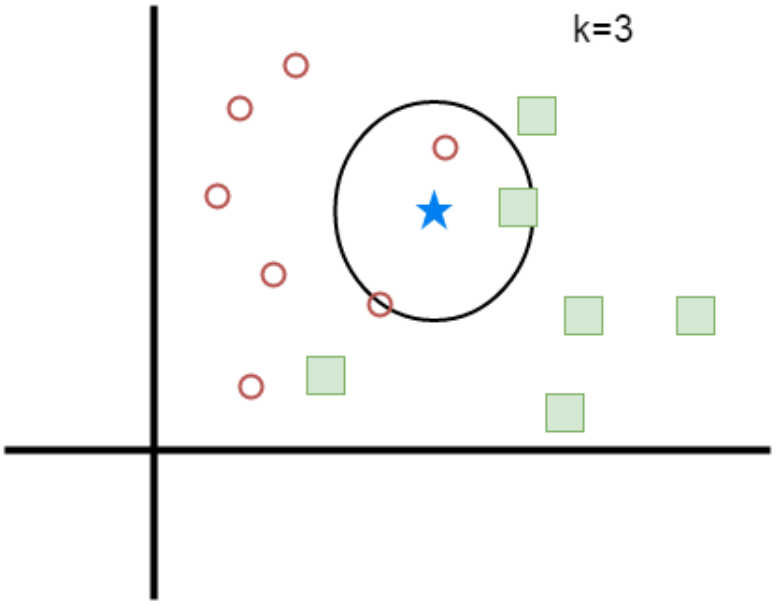
We can see now that a circle that has the blue star as its center was created, and it includes only the three red circle data points. So we can say with great confidence that the blue star should belong to the red circle family as his closest neighbors are all red circles. The choice of k was crucial in this example, and next on, we will study how to pick the best K for each case.

Here follows an example that can help you understand how important k is. In the following picture, we can see a similar example to the previous one but with more data points. Again, we are trying to find if the blue star is a red circle or a green square.



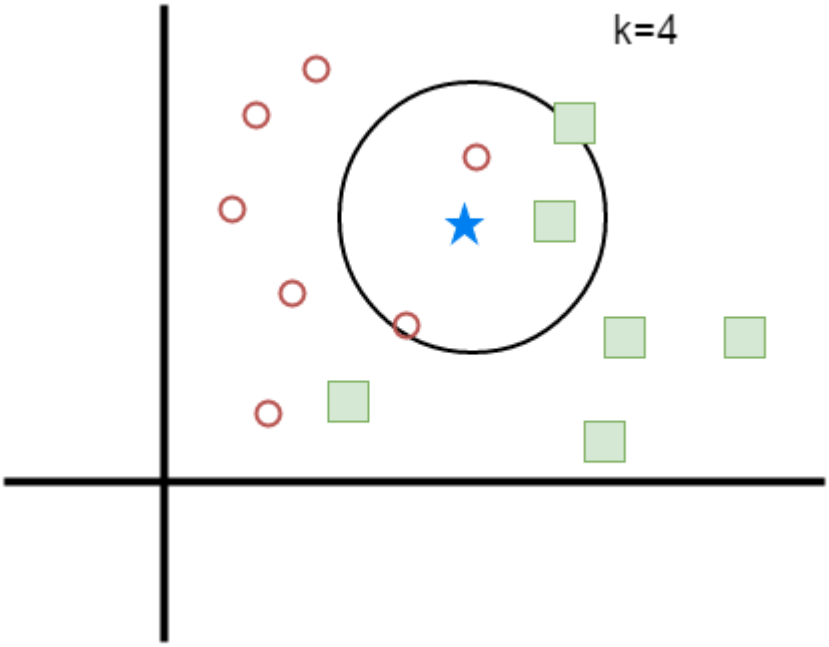
Picture 3.3: Knn Operation 3

First, we try to put the value of k=3. That means that we only consider the three nearest neighbors when we are trying to figure out the blue star. We can see that if k=3, then the blue star will probably be a red circle.



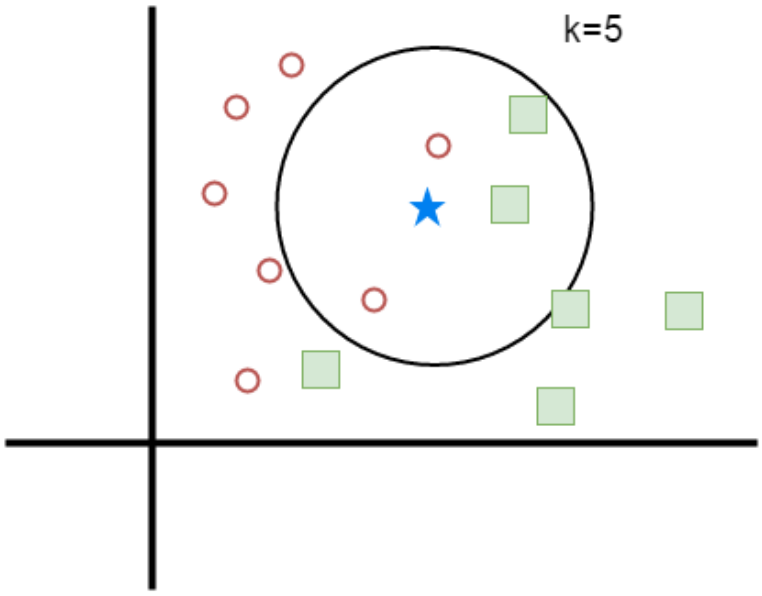
Picture 3.4: Knn Operation 4

If we change the value of k to be equal to 4, we can currently see that we can't decide what the blue star is, because now we take into account four neighbors, and we can see that the four nearest neighbors of the blue star are two red circles and two green squares.



Picture 3.5: Knn Operation 5

Last, if we change the value of k to be equal to 5, we can see that now the blue star is definitely a green square as we have three green square neighbors and two red cirle neighbors.



Picture 3.6: Knn Operation 6

In theory, the more neighbors we pick (high k), the better, but in order to do that you need to find a way to determine how their preferences will be related to the preferences of the active user. This of course, will generate a lot of noise as it's virtually impossible to do in a big database with a lot of dissimilar neighbors.

So a normal number for k is something between 25-100 neighbors or the square root of the number of data we have, which is what we used in this dissertation. If you have fewer neighbors, then it's easier to focus on the most similar neighbors and improve accuracy by eliminating noise. On the other hand, it will be much harder to find related users, so the number of items that you will be recommended on the final phase of the recommendation process will be decreased or be inaccurate.

3.6 Calculating Similarity Scores

After collecting all the different data points in knn you need a way to determine how similar those data points are. In order to do this, you need to have a similarity score between the different points, something that determines their similarity. There are many ways to do this, but the most famous ones are the Euclidean distance and Pearson correlation. In this dissertation, we picked Pearson correlation.

3.6.1 Why we choose Pearson correlation?

Since we are working on a movie recommender that has all the movies rated on a scale of 1 to 5, we picked Pearson correlation because it's the most appropriate one to use for measurements that belong in an interval scale. The basic advantage of this correlation is that it corrects for rating inflation, meaning that if a User tends to give lower ratings than another user, but their tastes still fit, then using Pearson correlation, they are still supposed to have similar preferences. That means that if the difference between their ratings is consistent, then there is still a correlation between those users in contrast with the Euclidean distance where the two users will not be correlated because one is harsher than the other.

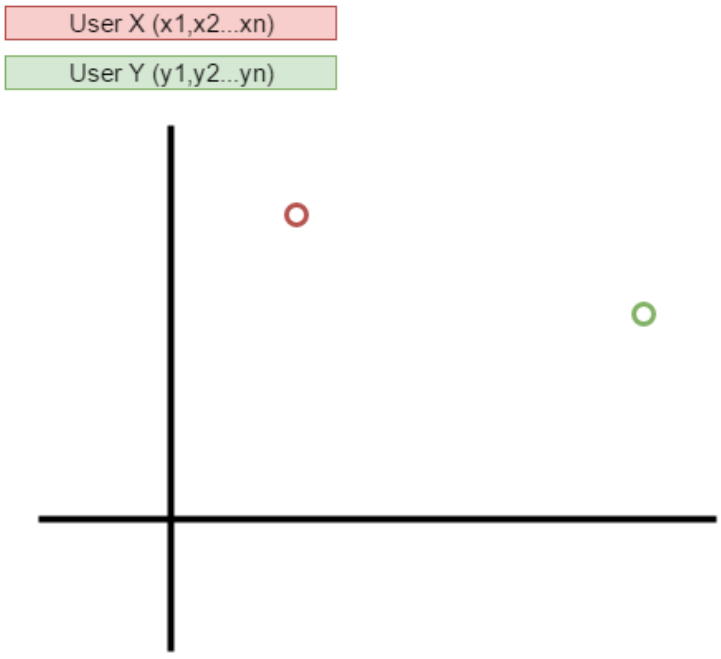
3.6.2 How Pearson correlation Works

As we mentioned before, Pearson Correlation is used as a similarity metric which tells us how similar users are in our movie recommender. It helps us find the nearest neighbors when we already have represented the user vectors as data points.

So given any two variables, the correlation is a measure of how similar those variables are or how similar the changes in those variables are. The Pearson correlation is nothing but the correlation that you will normally measure when you are trying to do something like a linear regression.

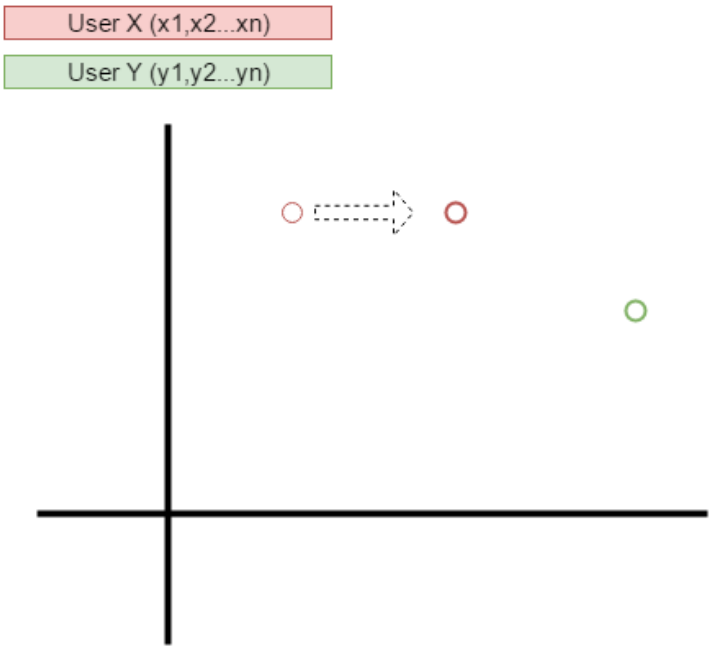
So, having in mind that the vectors are the user's ratings for different movies, each user might have a certain bias. As we said before, some users might rate some movies highly and some other users might have a general tendency to rate everything low. Pearson correlation helps to account that by normalizing each user's rating by their average rating.

Let's say we have two users, the red circle vector represents the user X and the green circle vector represents the user Y. User X has an average rating which will be the mean of rating x1, x2 to xn and also User Y will have his own average rating that will be the mean of y1, y2 to yn ratings.



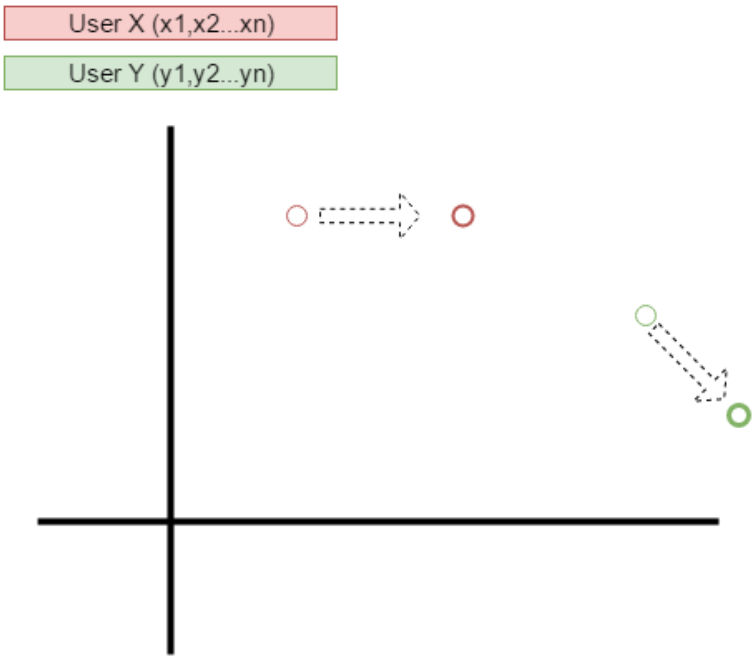
Picture 3.7: Pearson Correlation Operation 1

Now we take user X and we shift him by his mean, meaning that each number x1,x2 to xn is adjusted by the user's mean rating. This is what we called normalization because now this will give us a new point where the user will be normalized by his average rating. If $\bar{x}$ is the average rating of user x, the new tuple of the user will be x1-$\bar{x}$, x2-$\bar{x}$, x3-$\bar{x}$ and so on.



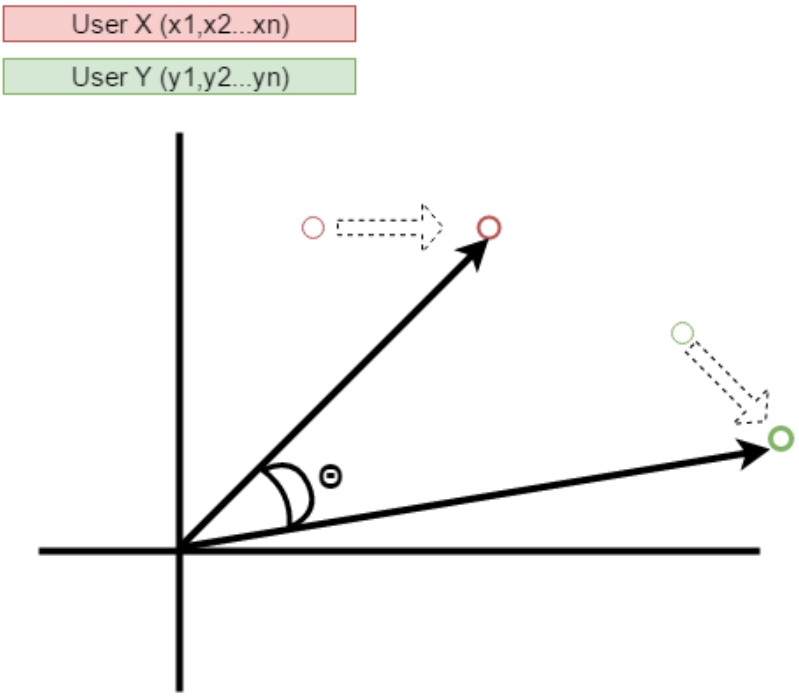
Picture 3.8: Pearson Correlation Operation 2

Now we do the same thing with user Y, meaning that we shift him based on his average rating. So again each number in the new tuple will be y1-$\bar{y}$, y2-$\bar{y}$, y3-$\bar{y}$ and so on, where $\bar{y}$ is the mean rating of all the movies that users Y has rated.



Picture 3.9: Pearson Correlation Operation 3

So currently we have adjusted user X and user Y to be represented such as, some of the bias that was preexisted has now been removed, and we managed to do that by normalizing for the average rating. The cosine similarity between these two new vectors is the Pearson correlation.



Picture 3.10: Pearson Correlation Operation 4

The formula for the Pearson Correlation is the following :

$$\text{Corr(x,y)} = \frac{\Sigma_{\boldsymbol{i}}(\boldsymbol{xi}-\bar{\boldsymbol{x}})(\boldsymbol{yi}-\bar{\boldsymbol{y}})}{\sqrt{\Sigma(\boldsymbol{xi}-\bar{\boldsymbol{x}})^{\boldsymbol{2}}}\sqrt{\Sigma(\boldsymbol{yi}-\bar{\boldsymbol{y}})^{\boldsymbol{2}}}}$$

$$= \frac{(x-\bar{x},\ y-\bar{y})}{\|x-\bar{x}\|\|y-\bar{y}\|}$$

$$=\text{CosSim}(x-\bar{x},\ y-\bar{y})$$

Equation 3.1: Pearson Correlation 2

As we can see again the pearson correlation is nothing more than the cosine similarity of $x-\bar{x}\ \ and\ \ y-\bar{y}$. As we said again, $\bar{x}$ represents the mean of all the ratings that the user X has given and $\bar{y}$ represents the mean of all the ratings the user Y has given.

3.7 Predict Rating Formula

Once we find the k-nearest neighbors of the active user, we use those neighbors to find the rating that the active user will give to any particular product. The formula used for this is the following:

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u\in U}(r_{u,i} - \bar{r}_u)\ .\ w_{a,u}}{\sum_{u\in U}|w_{a,u}|}$$

Equation 3.2:Predict Rating Formula 1

Here follows the explanation of the predict rating formula and how it works.

First, let's say that we have to predict the rating of the Active user *a* for product *i*. So $p_{a,i}$, is the predicted rating for the active user for product *i* . We first start with what the average rating of the active user *a* is for any product. So, if we have no information about what the active user's rating for a product would be, we just pick the average rating that the neighbors have given to other products ($\bar{r}_a$) .

Now we have some information, we have the nearest neighbors for the active user and we have their ratings for the product *i*. So we have to incorporate this into the prediction for the active user's rating. Then we have a summation over all the users in this set of nearest neighbors of the active user *a*. So for each of those neighbors we are computing some numbers and then picking the summation of that number ($\sum_{u \in U}$). As we might expect, that number has to do with the rating of that nearest neighbor for the product *i*.

So given a user *u*, in the set of nearest neighbors of the active user, $r_{u,i}$ is the rating of this user for the product *i* . Important think to notice is that we don't just directly use the user's rating for the product *i* , we are also adjusting it by that particular user's mean average rating $\bar{r}_u$. So we use $(r_{u,i} - \bar{r}_u)$ for our calculations.

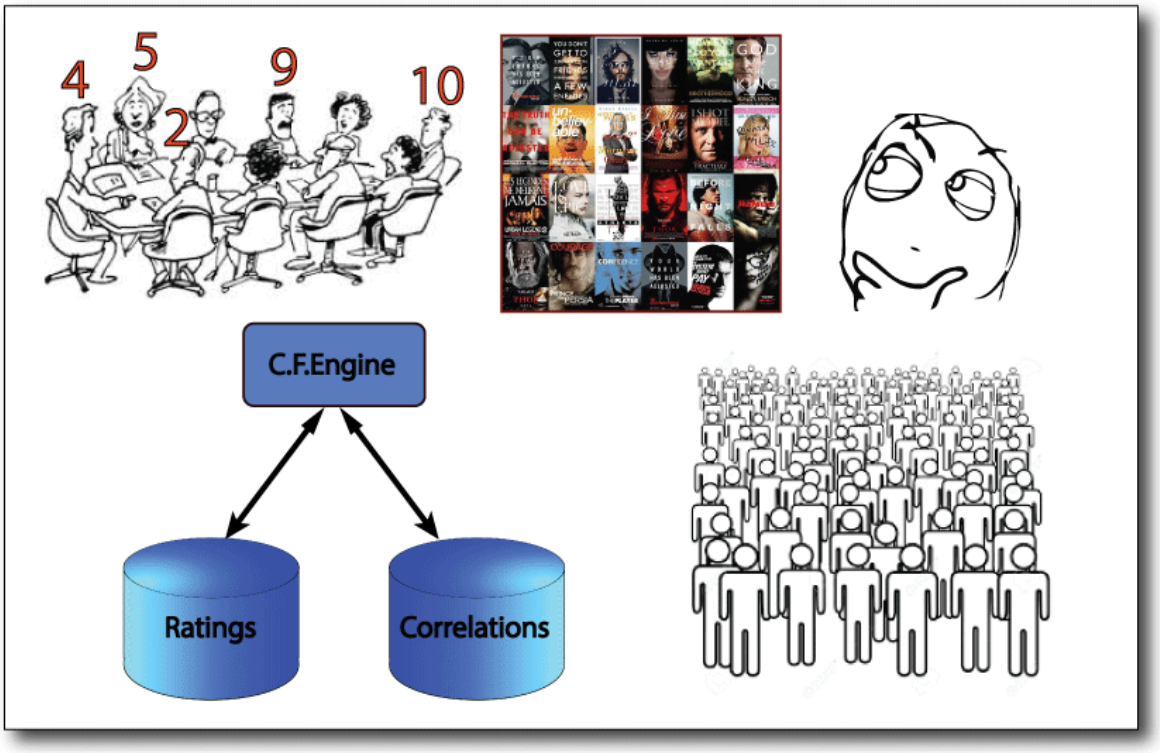
Last and most important step is to multiply this summation with $w_{a,u}$ which is a weight for that rating. This means that $w_{a,u}$ is the similarity between user *u* and the active user *a*, meaning in our case, the value of the Pearson correlation calculated for those 2 users.

In any case, what we need to remember is that the predicted rating is the weighted average rating of the nearest neighbors of a particular user.

3.8 How a Typical Movie Recommender System Works

After explaining the different concepts and vocabulary used in Recommender systems and particularly the ones using Collaborative filtering, we can proceed in the more detailed approach used in Movie Recommender systems.

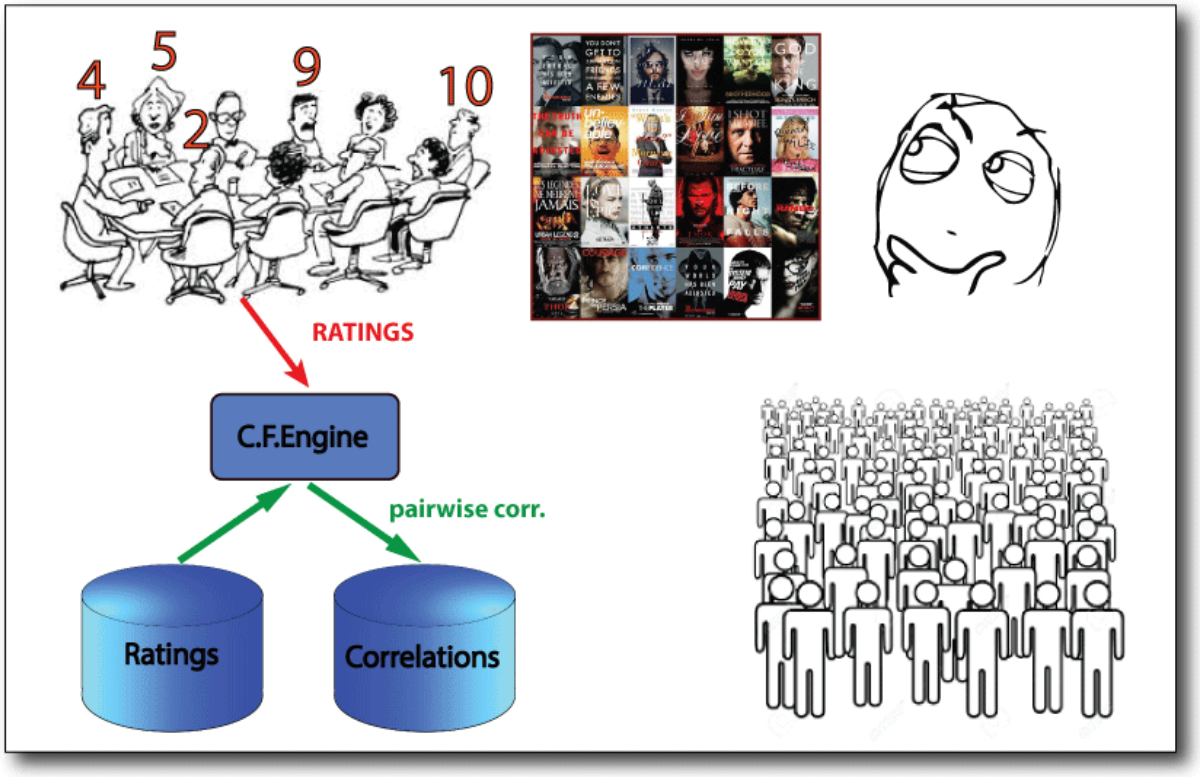
So, the idea of a Movie Recommender system is that different people rate movies in order for the system to understand their taste and return useful recommendations. In the following Picture we can see the active user and the different other users registered in the system.



Picture 3.11: Recommender System Operation 1

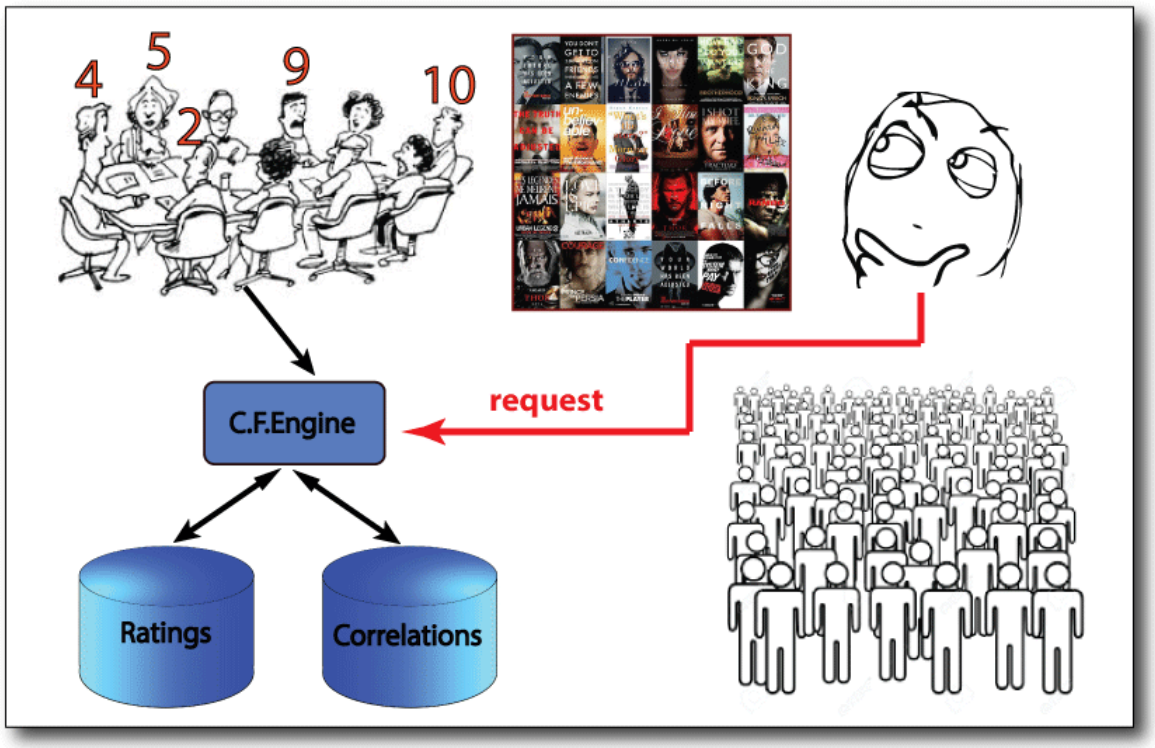
These registered users have already provided to the system various ratings for various movies. We assume that these ratings are in the range of 1-5, and they are pure integer numbers, like the way users rate in our 50/50 Movie Recommendation system. In the next picture, we can see that these ratings get through the collaborative filtering engine, after they are stored within the systems database, and they then used to make user correlations, meaning, finding and grouping users who share the shame taste.

This correlation, which we call pairwise correlation, is represented by a number which is close to 1 if two users share similar tastes and close to -1 if they don't. If the pairwise correlation is close to 0, then this means that they sometimes share the same interests and sometimes not. One really important factor is to also count the pieces of data these correlations are based on because that way we will be able to compute the overall pairwise correlation of two users and not just their correlation on one simple movie.



Picture 3.12: Recommender System Operation 2

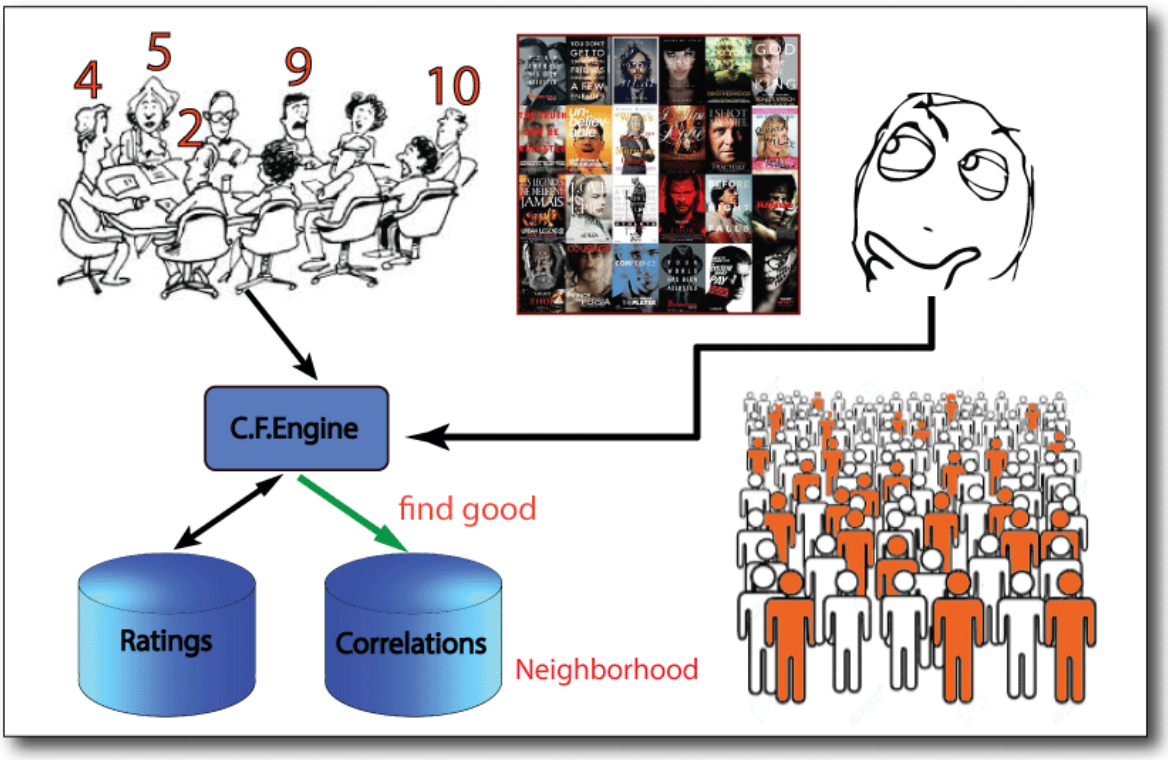
After all these calculations are done, it’s time for the active user to make a request. That usually translates into asking a recommendation from the recommendation system. In our Movie recommender system it means “suggest a list of movies that I will enjoy."



Picture 3.13: Recommender System Operation 3

Now, we can see on the next picture how collaboration filtering engine takes action. When the request from the active user comes in, the system uses the correlations that were calculated previously to find the nearest neighbors (users) that have similar tastes with our active user. This is the K-nearest neighbor method and is the one we used for this project. So, the people in red are actually the users who have the highest correlation with the active user, meaning that the correlation score is close to 1 as we mentioned earlier.

Final step is to have those strongly correlated users ratings combined by our collaborative filtering engine and suggest movies to the active user.



Picture 3.14: Recommender System Operation 4

3.9 Understanding The Computations

Here follows a really simple example of how a small movie recommender system works and how the calculations and computations are made to make the final movie recommendations. In the next table, we can see a rating matrix with 7 different users and their ratings for 6 different movies. What is our main goal is to predict if the active user will enjoy more Trainspotting or Liar Liar and recommend it to him.

Table 3.2: Computations 1

	Titanic	Alien	Terminator	Forrest Gump	Trainspotting	Liar liar
Active User	4	1	2	4		
John	1	5	4		5	
Peter	1	1	1	1	1	1
Joe	4	1		3		
Mary	1	3	1	3		1
Bill	5	1				5
Jenny	4		1		1	

So first, as we previously said, we are looking for users who share similar tastes to the active user. One of those users in our example is Jenny, as she gave Titanic the same rating as the active user and Terminator a pretty close one too.

Table 3.3: Computations 2

	Titanic	Alien	Terminator	Forrest Gump	Trainspottinq	Liar liar
Active User	4	1	*2*	4		
John	1	5	4		5	
Peter	1	1	1	1	1	1
Joe	4	1		3		
Mary	1	3	1	3		1
Bill	5	1				5
Jenny	4		1		1	

Next, we can also notice that the active user and John don't share the same taste in movies which even that might be helpful if used in a reversed sense.

Table 3.4: Computations 3

	Titanic	Alien	Terminator	Forrest Gump	Trainspottinq	Liar liar
Active User	4	1	*2*	4		
John	1	5	4		5	
Peter	1	1	1	1	1	1
Joe	4	1		3		
Mary	1	3	1	3		1
Bill	5	1				5
Jenny	4		1		1	

An interesting situation is Joe and the active user. Joe share really similar tastes with the active user but he is not going to be helpful in the recommender system as he hasn't seen any of the two movies we are interested in (Trainspotting, Liar Liar).

Table 3.5: Computations 4

	Titanic	Alien	Terminator	Forrest Gump	Trainspottinq	Liar liar
Active User	4	1	*2*	4	?	?
John	1	5	4		5	
Peter	1	1	1	1	1	1
Joe	4	1		3	?	?
Mary	1	3	1	3		1
Bill	5	1				5
Jenny	4		1		1	

So then it's time for the recommender to make a rating prediction for the movie Trainspotting for the active user. The two users who are the most important are John and Jenny. Jenny has almost similar tastes to the active user, and she doesn't like the movie and John who has almost the complete different taste, likes the movie a lot. Those two facts can lead to the conclusion that the active user won't probably like Trainspotting.

Table 3.6: Computations 5

	Titanic	Alien	Terminator	Forrest Gump	Trainspottinq	Liar liar
Active User	4	1	*2*	4	?	?
John	1	5	4		5	
Peter	1	1	1	1	1	1
Joe	4	1		3		
Mary	1	3	1	3		1
Bill	5	1				5
Jenny	4		1		1	

Now we have to follow the same procedure for the movie Liar Liar but the only safe evidence we have is that Bill who has similar taste to the active user, gave it an excellent rating. That means that the recommender system would most likely pick Liar Liar to recommender to the active user.

Table 3.7: Computations 6

	Titanic	Alien	Terminator	Forrest Gump	Trainspottinq	Liar liar
Active User	4	1	*2*	4	?	?
John	1	5	4		5	
Peter	1	1	1	1	1	1
Joe	4	1		3		
Mary	1	3	1	3		1
Bill	5	1				5
Jenny	4		1		1	

4 HUMAN FACTOR AND THE BIG FIVE PERSONALITY TEST

4.1 Human Factor

Human factor is the procedure of designing systems and products, which take into special account the interaction between them and the people who will use them. This practice is widely used in industrial design and engineering and has seen contributions from psychology, physiology and anthropometry. Human factor is α way to optimize the best user experience and overall system performance. As mentioned in the International Ergonomics Association, the official definition of human factor is the following:

"Ergonomics (or human factors) is the scientific discipline concerned with the understanding of interactions among humans and other elements of a system, and the profession that applies theory, principles, data and methods to design in order to optimize human well-being and overall system performance."

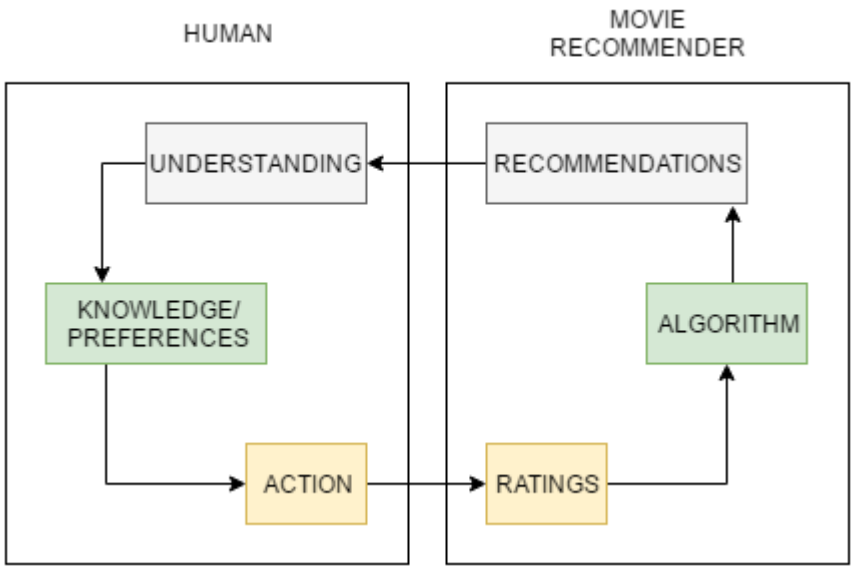
Also, we find another definition in the following quote from the U.S. National Academy of Engineering's:

"Engineers and engineering will seek to optimize the benefits derived from a unified appreciation of the physical, psychological and emotional interactions between information technology and humans. As engineers seek to create products to aid physical and other activities, the strong research base in physiology, ergonomics and human interactions with computers will expand to include cognition, the processing of information, and physiological responses to electrical, mechanical and optimal stimulation"

In this dissertation, we implement the human factor in the recommender systems and try to maximize the user-system interaction and make the user trust the recommender more. Our way of doing that is by incorporating the Big Five personality test which helps the recommender understand better each user and filter the recommendations based on the active user's personality.

Every system has a goal. The goal of our Movie Recommender is to provide the best movie recommendations for every different user, based on his personality and his previous movie ratings. To achieve this goal, both the system and the user must operate in a correct and meaningful way. The efficiency of the system is totally dependable on both the performance of the recommendation engine and the performance of the user. That means that if the recommendation engine is poorly designed and provides bad recommendations, or it demands too much effort from the user to operate, the user will lose interest and trust, and the system will fail. Likewise, if the user doesn't trust the system from the beginning and provides false ratings or doesn't fulfill the Big Five personality test, then the system again will fail.

In the following picture, we can see a clear version of the Human- Movie recommender interaction. We can see that both are a combination of different subsystems used for input, processing and output. Below we will examine and explain the complete procedure that the human must follow, so that the system will operate correctly.



Picture 4.1 Human-Movie Recommender Interaction

First, the user must act and provide ratings to the recommender. Without the action from the user, the movie recommender won't have an input and won't be able to operate. The central algorithm of the recommender system is completely analogous to the knowledge and preferences of the user. The algorithm of the movie recommender works based on other user's preferences and also needs the active user's preferences to operate. Finally, the recommender produces recommendations and the user must have an understanding for those recommendations so to trust the recommender and use it again in the future.

Furthermore, the following diagram is pretty helpful to describe the exact procedures we had to make to complete this dissertation project. First, in the movie recommender system, we researched the cold start problem so that the user will provide the best ratings to the recommender system. A big plus is the Big Five personality test but currently there is no need to include it in the diagram. Second, we designed a fast and efficient algorithm that produces recommendations based upon the user nearest neighbors. Last but not least, we presented the final recommendations.

So a good recommendation system has all the above characteristics, but the human factor can contribute massively to its usability. As we mentioned earlier, our way of adding the human factor into this movie recommender system is by introducing the human personality into the equation. This addition makes it easier for the system to communicate and understand the user's needs. There is always a tradeoff between usability and flexibility but in our case, we believe that the user must pass all the appropriate procedures so that the recommender gets the most information.

Last but not least, there are for sure some drawbacks in the addition of the Personality test into the movie recommender system, and they correlate a lot with some general weaknesses of the Human Factor and ergonomics methods. First, with the addition of the Big Five personality test, it takes more time for the user to sign in and more resources of information than the usual sign in methods used in other movie recommender systems. Second, extra planning and research were needed to make the algorithm take into account the personality scores of each user. And finally, there are many methods to detect the active user's personality and there is no correct way to do it, so it's completely subjective and depends only to the judge of the programmer.

4.2 Background On The Big Five Personality Test

The Big Five personality test is considered to be one of the most accepted ways to identify the personality of a person in modern academic research and psychology. It's actually the result of a statistical study of the answers to specific questions (personality items) people from Western Europe and America provided. The researchers then can use this data to find the best ways to summarize an individual.

Although, there were many personality variables, Extraversion, Agreeableness, Conscientiousness, Neuroticism and Openness to Experience where the ones that stood out the most and the ones that are being used in the Big Five Personality Test.

4.3 What Are The Big Five traits?

The big five personality test helps you calculate your score for five different personality traits, which are the following as stated [**14**]:

• **Extroversion (E) is** the trait of seeking fulfillment from sources outside the self or in community. High scorers tend to be very social while low scorers prefer to work on their projects alone.

• **Agreeableness (A)** reflects how much individuals adjust their behavior to suit others. High scorers are typically polite and likeable people. Low scorers tend to 'tell it like it is'.

• **Conscientiousness (C)** is the personality trait of being honest and hardworking. High scorers tend to follow rules and prefer clean homes. Low scorers may be messy and cheat others.

• **Neuroticism (N)** is the personality trait of being emotional.

• **Openness to Experience (O)** is the personality trait of seeking new experience and intellectual pursuits. High scores may day dream a lot. Low scorers may be very down to earth.

4.4 Some Issues With The Big Five Personality Test

It is important to point out that there are many aspects of the human personality and psychology that the Big Five Personality Test can't portray. When we are talking about "traits" of a person, we are talking about something much narrower and conceptually distinct. The Big Five traits are based on empirical and statistical studies, and it's not a theory of personality. This is why we should consider the results of this test to be a partial and general description of the user's personality but on the other hand, that would be more than enough for our system, as it minimized the user's effort and provides the general characteristics of the user.

4.5 Further Analysis Of Each Trait

Extraversion

Extraversion is the trait that describes a person being an extrovert. This person is full of energy and enthusiasm. They tend to take actions and handle opportunities with excitement. They like being the center of attention and starting conversations.

The opposite of extraversion are people who are introverts. They are quiet and don't like to be around people, but that doesn't mean they are shy or depressed. They just prefer to be alone.

Agreeableness

People with high Agreeableness tend to be friendly, helpful and ready to compromise their needs for others. They are really optimistic and believe that most people are trustworthy and honest. They also tend to be more popular

Users with a low level of Agreeableness are usually selfish and believe that their needs are above everything else. They usually don't help others and are unfriendly and sometimes rude. On the other hand, they can be really good in making the correct decisions in tough times.

Conscientiousness

Conscientiousness is the trait that has to do with how one person controls his instincts and his impulses. A person with high level of conscientiousness is a person who thinks about the future and the consequences of his actions. High conscientiousness usually means also high intelligence, being able to organize and plan goals and being also able to sacrifice short joys for a greater future. You can consider a person with high conscientiousness as both wise and cautious at the same time. Conscientious people though can be workaholics and perfectionist.

People with low conscientiousness are usually people who can be more spontaneous and fun as they act based on their instincts and impulses than logic. This though may lead in many troubles like being antisocial or even expressing a destructive behavior. They prefer immediate results and rewards and don't have the patience to follow a plan for a better future. They are unreliable and they usually lack ambition.

Neuroticism

Neuroticism is the tendency that each person has to experience negative feelings and the inability to handle the normal demands of life. People with high neuroticism can experience depression, anxiety attacks, bursts of anger and generally everything that has to do with extreme reactions to emotions. They are most of the time in bad mood, and they can't think and operate clearly, making it really difficult to take the correct decisions as they will be usually under a lot of stress.

Of course, people who score low on neuroticism are less sensitive in stress and can control their emotions better. Negative feelings are not something that concerns them and they can stay calm and relaxed even in difficult situations. Although that doesn't imply the opposite, meaning that they are full of positive feelings, as this is a characteristic of mostly people with high Extraversion

Openness to experience.

Openness to Experience is what separates people with big imagination and people who are generally down to earth. People with high Openness are usually artistic and curious and more connected to their feelings. This trait tends to be a characteristic of people that prefer individualism and culture.

People who score low on openness to experience tends to have common sense and narrow interests. They are straightforward people and may even consider art, science and music something useless and with no point. They are generally conservative and resist to any change, positive or negative.

4.6 Big-Five Traits Markers

Here follows a list of the questions in the Big Five Personality test and the corresponding Personality trait that it affects. The + symbol means that the answer on these specific questions raises the score of the specific personality trait and the – symbol that it lowers it

Extraversion

+	Am the life of the party. Feel comfortable around people. Start conversations. Talk to a lot of different people at parties. Don't mind being the center of attention.
–	Don't talk a lot. Keep in the background. Have little to say. Don't like to draw attention to myself. Am quiet around strangers.

Agreeableness

\+

Am interested in people.
Sympathize with others' feelings.
Have a soft heart.
Take time out for others.
Feel others' emotions.
Make people feel at ease.

−

Am not really interested in others.
Insult people.
Am not interested in other people's problems.
Feel little concern for others.

Conscientiousness

\+

Am always prepared.
Pay attention to details.
Get chores done right away.
Like order.
Follow a schedule.
Am exacting in my work.

−

Leave my belongings around.
Make a mess of things.
Often forget to put things back in their proper place.
Shirk my duties.

Neuroticism

+	Am relaxed most of the time.
	Seldom feel blue.
−	Get stressed out easily.
	Worry about things.
	Am easily disturbed.
	Get upset easily.
	Change my mood a lot.
	Have frequent mood swings.
	Get irritated easily.
	Often feel blue.

Oppenness

+	Have a rich vocabulary.
	Have a vivid imagination.
	Have excellent ideas.
	Am quick to understand things.
	Use difficult words.
	Spend time reflecting on things.
	Am full of ideas.
−	Have difficulty understanding abstract ideas.
	Am not interested in abstract ideas.
	Do not have a good imagination.

4.7 Big Five Traits And Human Characteristics

The following table will help understand and relate some basic human characteristics with the Big Five Personality traits. We also provide the opposite trait so that it will be easier to get an idea about what approximately is each trait.

Table 4.1: Connecting Traits with Human Characteristics

Big Five Traits	Opposite trait	Facets
Extraversion	**Introversion**	Gregariousness (sociable) Assertiveness (forceful) Activity (energetic) Excitement-seeking (adventurous) Positive emotions (enthusiastic) Warmth (outgoing)
Openness	**Closedness to experience**	Ideas (curious) Fantasy (imaginative) Aesthetics (artistic) Actions (wide interests) Feelings (excitable) Values (unconventional)
Conscientiousness	**Lack of direction**	Competence (efficient) Order (organized) Dutifulness (not careless) Achieve merit striving (thorough) Self-discipline (not lazy) Deliberation (not impulsive)
Neuroticism	**Emotional stability**	Anxiety (tense) Angry hostility (irritable) Depression (not contented) Self-consciousness (shy) Impulsiveness (moody) Vulnerability (not self-confident)
Agreeableness	**Antagonism**	Trust (forgiving) Straightforwardness (not demanding) Altruism (warm) Compliance (not stubborn) Modesty (not show-off) Tender-mindedness (sympathetic)

4.8 Big Five Personality Test Questions

Here are the questions that the user has to answer so that we can understand the way he acts and how his personality is structured. The user must answer these questions and describe him as he is now and not as he wishes to be in the future. In order to do that, the user can make a comparison between him and other people similar to his age and sex. As we have previously mentioned, the user has to score the questions with an answer from 1-5 where,

- 1=disagree,
- 2=slightly disagree,
- 3=neutral,
- 4=slightly agree and
- 5=agree.

Before all the questions we have the prefix " I " .

1. Am the life of the party.
2. Feel little concern for others.
3. Am always prepared.
4. Get stressed out easily.
5. Have a rich vocabulary.
6. Don't talk a lot.
7. Am interested in people.
8. Leave my belongings around.
9. Am relaxed most of the time.
10. Have difficulty understanding abstract ideas.
11. Feel comfortable around people.
12. Insult people.
13. Pay attention to details.
14. Worry about things.
15. Have a vivid imagination.
16. Keep in the background.
17. Sympathize with others' feelings.

18. Make a mess of things.
19. Seldom feel blue.
20. Am not interested in abstract ideas.
21. Start conversations.
22. Am not interested in other people's problems.
23. Get chores done right away.
24. Am easily disturbed.
25. Have excellent ideas.
26. Have little to say.
27. Have a soft heart.
28. Often forget to put things back in their proper place.
29. Get upset easily.
30. Do not have a good imagination.
31. Talk to a lot of different people at parties.
32. Am not really interested in others.
33. Like order.
34. Change my mood a lot.
35. Am quick to understand things.
36. Don't like to draw attention to myself.
37. Take time out for others.
38. Shirk my duties.
39. Have frequent mood swings.
40. Use difficult words.
41. Don't mind being the center of attention.
42. Feel others' emotions.
43. Follow a schedule.
44. Get irritated easily.
45. Spend time reflecting on things.
46. Am quiet around strangers.

47. Make people feel at ease.

48. Am exacting in my work.

49. Often feel blue.

50. Am full of ideas.

4.9 Scoring In Big Five Test

When all the questions are answered, the five personality traits are being calculated with the following formulas, where the number in brackets is the corresponding question and next, you put the number on the range of 1-5 you previously put in an answer:

Extraversion = 20 + (1) ___ - (6) ___ + (11) ___ - (16) ___ + (21) ___ - (26) ___ + (31) ___ - (36) ___ + (41) ___ - (46) ___ = _____

Agreeableness = 14 - (2) ___ + (7) ___ - (12) ___ + (17) ___ - (22) ___ + (27) ___ - (32) ___ + (37) ___ + (42) ___ + (47) ___ = _____

Conscientiousness = 14 + (3) ___ - (8) ___ + (13) ___ - (18) ___ + (23) ___ - (28) ___ + (33) ___ - (38) ___ + (43) ___ + (48) ___ = _____

Neuroticism = 38 - (4) ___ + (9) ___ - (14) ___ + (19) ___ - (24) ___ - (29) ___ - (34) ___ - (39) ___ - (44) ___ - (49) ___ = _____

Openness = 8 + (5) ___ - (10) ___ + (15) ___ - (20) ___ + (25) ___ - (30) ___ + (35) ___ + (40) ___ + (45) ___ + (50) ___ = _____

An important note here is to mention that in any kind of personality test, there is no actual average value. “Norms” are really misleading and generally should be avoided because one person’s score could never be a representative value or subset in a population. The correct way to find the average values for each trait is to find the average value of each trait on the sample that was used, meaning, the people we evaluated our recommender.

Here follows the average values of each trait based on the 30 different users we tested our recommender. Every value below this score is considered a low value and every value above, a high value.

Extraversion: 3.45

Agreeableness: 3.96

Conscientiousness: 3.46

Neuroticism: 3.128

Openness: 3.86

On the next page we present again some human characteristics based on if the user has scored high or low on a specific trait.

Neuroticism

Low	High
Stable	Tense
Calm	Anxious
Contented	Nervous
Unemotional	Moody
	Worrying
	Fearful

Agreeableness

Low	High
Cold	Sympathetic
Unfriendly	Kind
Unking	Appreciative
Cruel	Warm
Thankless	Generous
Fault-Finding	Trusting

Conscientiousness

Low	High
Careless	Organized
Irresponsible	Thorough
Forgetfull	Efficient
Frivolous	Responsible
Undependable	Reliable
Slipshot	Precise

Extraversion

Low	High
Quiet	Sociable
Reversed	Forceful
Shy	Energetic
Silent	Adventurous
Withdrawn	Enthusiastic
Retiring	Outgoing

Openness

Low	High
Narrow interests	Imaginative
Simple	Intelligent
Shallow	Original
Unintelligent	Insightful
	Artistic
	Clever

5 DESIGN OF THE 50/50 MOVIE RECOMMENDER

5.1 Overview

The movie recommender we designed is called the 50/50 Movie Recommender system.

As mentioned before, it's a Movie recommender that takes into account the personality of the active user. It is based on collaborative filtering techniques and the Knn algorithm with the addition of the Big Five Personality test, which filters the final recommendations based upon the user's genre preferences.

5.2 MovieLens Dataset

The dataset we used is the MovieLens 100k database. It's a stable benchmark dataset, developed and collected by the GroupLens Research Progject, that includes 100.000 ratings from 1000 users on 1700 movies. It was released in 1998. It's really important that all the users in the Database have rated at least 20 movies, which helps the Knn operate as it makes it easier to find neighbors to the active user.

The MovieLens Database includes the following files that were using in our system:

A. *U.data*: The dataset with 100.000 rating by 943 users on 1682 movies. All users and items are numbered consecutively from 1. There is also a timestamp tab which was removed as it was not needed. On the next table, the head of U.data is presented:

Table 5.1: U.data Head

	userid	itemId	rating
0	196	242	3
1	186	302	3
2	22	377	1
3	244	51	2
4	166	346	1

B. *U.item*: Information about the movies. This includes movie id, movie title, release date, video release date, Imdb URL and 19 fields that represents the different genres, with 1 indicating that the movie is of that genre. Movies can be in several genres. We deleted the release date, video release date and Imdb url. In addition, we deleted some genres so for the recommender to operate better. Explanation in chapter ().On the next table the head of U.item is presented:

Table 5.2: U.item Head

	Item-Id	title	Act-ion	Adve-nture	Ani-mation	Car-toon	Com-edy	Dra-ma	Film-Noir	Horror	Rom-ance	Scl-Fl	War
0	1	Toy Story (1995)	0	1	0	0	0	0	0	0	0	0	0
1	2	GoklenEye (1995)	1	0	0	0	0	0	0	0	0	0	0
2	3	Four Rooms (1995)	0	0	0	0	0	0	0	0	0	0	0
3	4	Get SHorty (1995)	1	0	0	0	0	0	0	0	0	0	0
4	5	Copycat (1995)	0	0	1	0	0	0	0	0	0	0	0

5.3 Extra Information Added

Apart from the MovieLens database, two more important tables were used to help the operation of our system:

A. *5 factor table*: This is a table we created that includes the information provided on (). We deleted the cult, foreign, independent, tragedy, parody genres because they are not included in the Movielens dataset. For further information about this table check 2.4.5.

Table 5.3: Five Factor Table

	OPE	CON	EXT	AGT	NEU
action	3.87	3.45	3.57	3.58	2.72
adventure	3.91	3.56	3.54	3.68	2.61
animation	4.04	3.22	3.26	3.35	3.02
cartoon	3.95	3.33	3.49	3.57	2.81
comedy	3.88	3.44	3.58	3.6	2.75
drama	3.99	3.43	3.66	3.6	2.86
film-noir	4.34	3.35	3.33	3.37	2.97
horror	3.9	3.38	3.52	3.47	2.91
romance	3.84	3.48	3.62	3.62	2.85
sci-fi	3.99	3.55	3.33	3.57	2.73
war	3.82	3.51	3.49	3.5	2.71

B. *Most rated movies table*: This is a table we created with Microsoft Excel. It includes the most rated movies from the Movielens Dataset in Descending order. It is used to help solve the cold start problem be providing some famous movies for the user to rate at the beginning of the recommendation process. Here follows the head of the table:

Table 5.4: Most rated movies Table

MOVIE NAME	MOVIE ID	NUMBER OF RATINGS
Star Wars (1977)	50	583
Contact (1997)	258	509
Fargo (1996)	100	508
Return of the Jedi (1983)	181	507
Liar Liar (1997)	294	485
English Patient. The (1996)	286	481
Scream (1996)	288	478
Toy Story (1995)	1	452
Air Force One (1997)	300	431
Independence Day (ID4) (1996)	121	429
Raiders of the Lost Ark (1981)	174	420
Godfather. The (1972)	127	413

5.4 Assumptions

It’s important to make some assumptions about the data and the recommender so to avoid technical problems, at least in this research stage. Here are the assumptions we made:

a) The items set do not change. Meaning that our database remains stable and no new movies are added.

b) The users set do not change. Meaning the database with the different users and their different ratings does not change. Even though the new user enters the system and passed his name, the addition of his record to the user database is only temporary.

c) There is no browse history. We don't take into account what the user browses, only what movies he has already rated.

d) Always a fixed number of movies are recommended at the end to the user.

e) The user doesn’t change his personality through time.

5.5 System Architecture

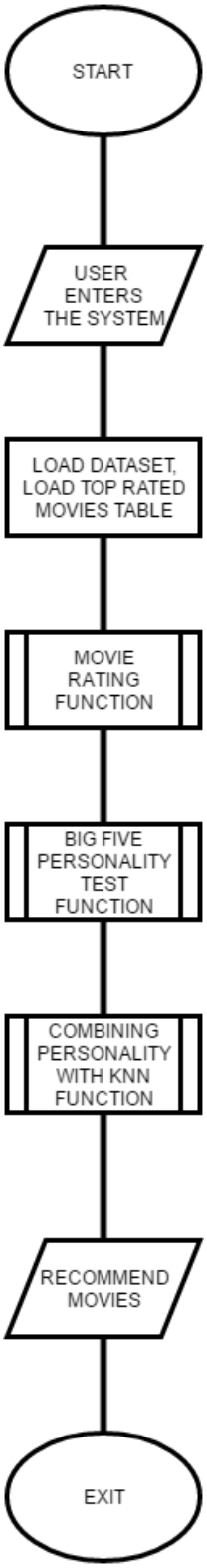
Our movie recommender consists of some different components and operations. Here follows the basic steps of how it operates:

- **Input** : Movielens database, 5 factor table, most rated movies table
- **Output** : Three sets of movie recommendations, one pure knn, one 50% knn 50 % personality, one 20% knn 80% personality
- **Steps of Algorithm** :

1. Start
2. User enters the system
3. Load Movielens dataset and 5factor and most rated movies tables
4. Active User Movie Rating Function
5. Active User Personality Test Function
6. Present the three different sets of movie recommendations
7. Stop

5.6 Main Flow Chart Explanation

On the next page we provide the *Main* flow chart of our recommender system.



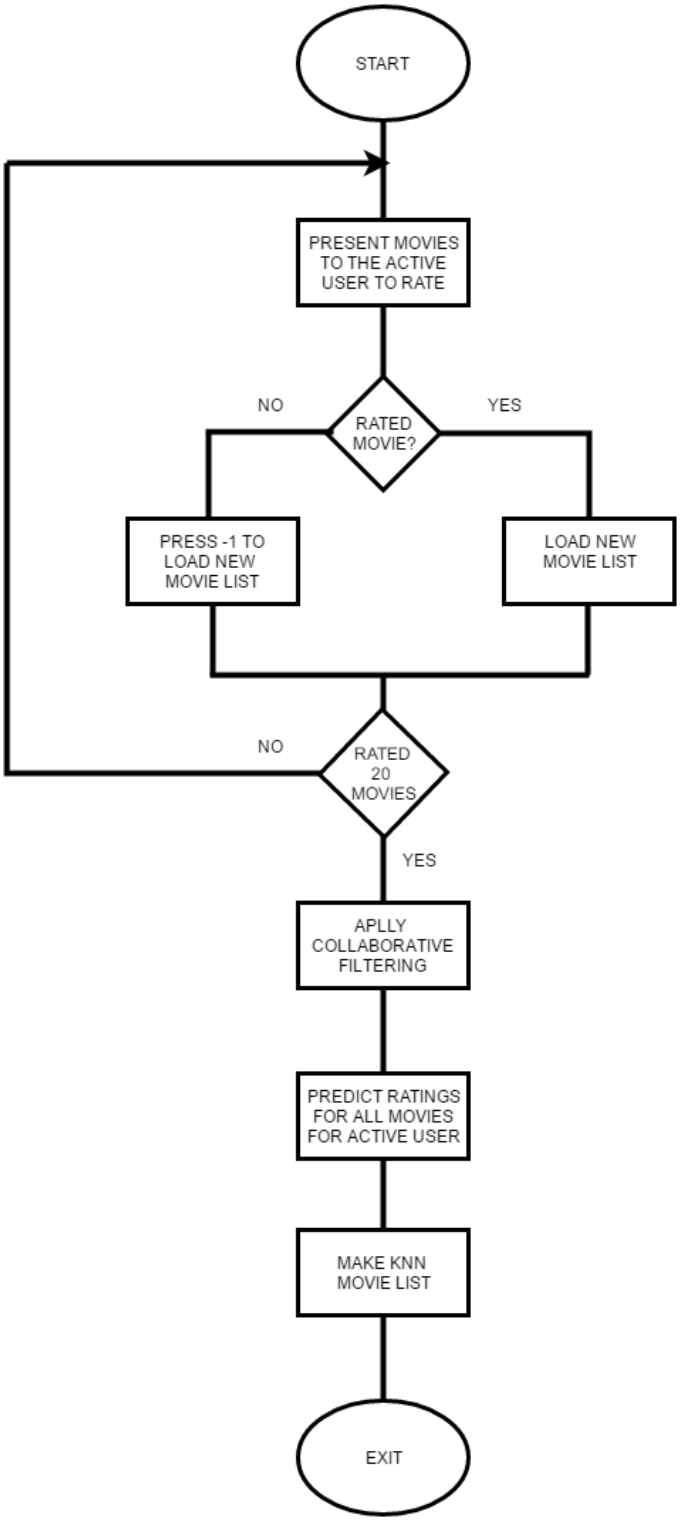
Picture 5.1:Main Flow Chart

So the user enters the system with the intention to have some movies recommended to him. Then we import the movieLens Dataset, and the Top rated movies table and proceed to the movie rating function.

Then a *Movie Rating function* is used to solve the cold start problem for the new user, as the system knows nothing about his movie preferences. Next, the user is asked to fulfill the big five questionnaire which is what the *Big Five Personality Function* actually is. Last function is the *Combining Personality with Knn Function,* which is the one that combines the Knn movie recommendations with the personality of the active user. Here follows some further analysis of each function and their corresponding flow chart.

5.7 Movie Rating Function Flow Chart Explanation

On the next page we provide the *Movie Rating Function* chart of our recommender system.



Picture 5.2:Movie Rating Function Flow Chart

So this function is basically used to solve the cold start problem, meaning that the system knows nothing about the active user and there needs to be a way for the user to provide information to the system.

So, first a list of 20 movies is presented to the user. Here we use the MostRatedMovies table, and we present one movie from this table and one random from the whole dataset alternately. That way, we avoid providing biased movies to rate and at the same time manage to suggest some famous and some not so well known movies. An example is presented on the following table.

Table 5.5:Movies Presented to the User

```
MOVIE LIST


↑MOVIE = 1/ 20
---------------------------------------------------------------------
        Star Wars (1977)    50
                          Faust (1994)  1367
                        Contact (1997)   258
   My Crazy Life (Mi vida loca) (1993)  1421
                          Fargo (1996)   100
               Princess Caraboo (1994)  1285
             Return of the Jedi (1983)   181
                   Passion Fish (1992)   972
                      Liar Liar (1997)   294
           My Own Private Idaho (1991)   537
           English Patient. The (1996)   286
   Bitter Sugar (Azucar Amargo) (1996)  1639
                         Scream (1996)   288
                          Giant (1956)   614
                      Toy Story (1995)     1
Maya Lin: A Strong Clear Vision (1994)   119
                  Air Force One (1997)   300
                Escape from L.A. (1996)   831
         Independence Day (ID4) (1996)   121
                          Patton (1970)   205
Put the movie ID of the movie you want to rate, or -1 to refresh list:
```

If the user cannot find any movie that he knows on the list, he just enters -1 and a new list is presented to him, following the same concept, one movie from the MostRatedMovies table and one random alternately.

This goes on until the user has rated 20 movies, which a pretty good number for the Collaborative algorithm to work. The system then uses collaborative techniques to find the K-nearest neighbors of the active user, meaning users that have a similar taste to him. The pairing is done as we said before, with the help of the Pearson Correlation and the following formulas.

$$\text{Corr}(x,y)= \frac{\Sigma_i (xi-\bar{x})(yi-\bar{y})}{\sqrt{\Sigma(xi-\bar{x})^2}\sqrt{\Sigma(yi-\bar{y})^2}}$$

Equation 5.1:Pearson Correlation 3

Next step is to predict the ratings the active user will give to all the movies he hasn't seen, based on his K- nearest neighbors. The function used for calculating the predicted rating is the following:

$$p_{a,i} = \bar{r}_a + \frac{\sum_{u \in U} \left(r_{u,i} - \bar{r}_u\right).w_{a,u}}{\sum_{u \in U} \left|w_{a,u}\right|}$$

Equation 5.2:Predict Rating Formula 2

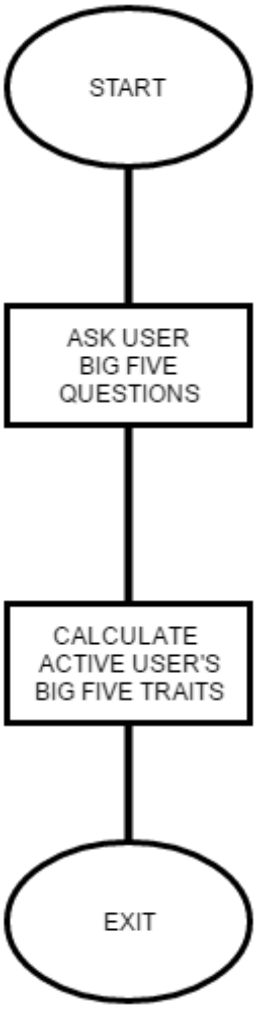
Last, a table with all the predicted ratings, plus the ones he provided at the beginning, is made. This is the most important list in our system as it's the basis in the application of the 50/50 and 80/20 calculations later.

Table 5.6: Predicted Ratings

itemId	Rating
1	7,29552
2	2,87429
3	2,87429
4	2,87429
5	2,87429
6	2,87429
7	-2.34085
8	2,87429
9	-1.42448
10	2,87429

5.8 Big Five Personality Test Function Flow Chart

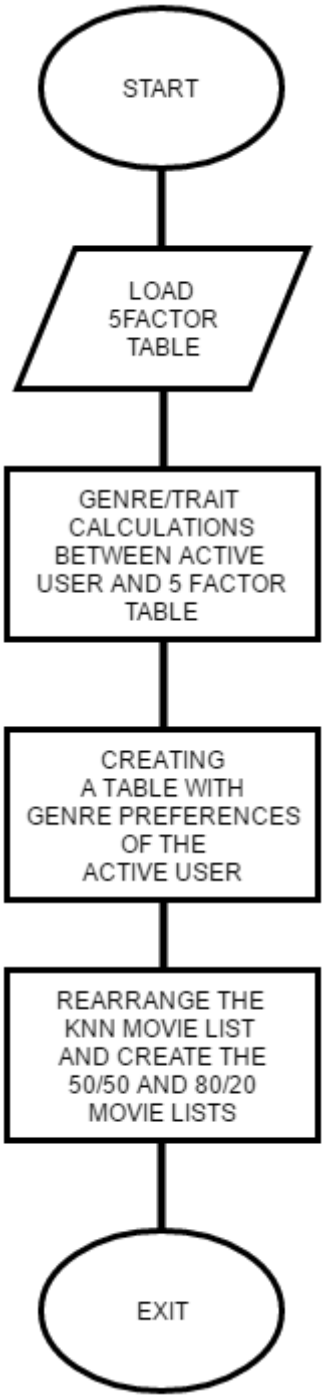
Here we provide the Big Five Personality Test function of our recommender system. It's just a function that presents to the user all the questions from the Personality Test. Then the system calculates the active user's big five traits score, based on the method explained on 4.9.



Picture 5.3:Big Five Test Function

5.9 Combining Personality With Knn Flow Chart

Here we provide the *Combining Personality With Knn* function of our recommender system. This is a function that we designed and implemented, and it's the main contribution of this Dissertation.



Picture 5.4:Combining Personality with Knn Function

So, first the 5factor table is loaded which, as we mentioned, is a table that includes the information provided on ().

Table 2.7: Correlation between Genre and Big Five Traits

	All users					
MOVIE GENRE	OPE	CON	EXT	AGR	NEU	#users
action	3.87	3.45	3.57	3.58	2.72	2488
adventure	3.91	3.56	3.54	3.68	2.61	179
animation	4.04	3.22	3.26	3.35	3.02	85
cartoon	3.95	3.33	3.49	3.57	2.81	957
comedy	3.88	3.44	3.58	3.60	2.75	3969
cult	4.27	3.10	3.45	3.40	3.16	38
drama	3.99	3.43	3.66	3.60	2.86	905
foreign	4.15	3.46	3.47	3.54	2.81	112
horror	3.90	3.38	3.52	3.47	2.91	2284
independent	4.31	3.59	3.51	3.55	2.69	104
neo-noir	4.34	3.35	3.33	3.37	2.97	92
parody	4.13	3.36	3.35	3.28	2.73	25
romance	3.84	3.48	3.62	3.62	2.85	776
science fiction	3.99	3.55	3.33	3.57	2.73	215
tragedy	4.40	3.34	3.27	3.52	3.11	26
war	3.82	3.51	3.49	3.50	2.71	148
	4.05	**3.41**	**3.46**	**3.51**	**2.84**	

What we do next is we take the score of the Big Five Traits of the active user, which we calculated previously on the *Big Five Personality Test* Function. Let's say that the active user has the following results:

Openness**: 3.87**

Conscientiousness**: 3.56**

Extraversion**: 3.53**

Agreeableness**: 4.7**

Neuroticism**: 2.89**

Based on these results, we can check the 5factor table and see at first sight which genres the active user prefers based on his personality.

Table 5.8:Genre Preferences of User

	All users				
MOVIE GENRE	OPE	CON	EXT	AGR	NEU
action	3.87	3.45	3.57	3.58	2.72
adventure	3.91	3.56	3.54	3.68	2.61
animation	4.04	3.22	3.26	3.35	3.02
cartoon	3.95	3.33	3.49	3.57	2.81
comedy	3.88	3.44	3.58	3.60	2.75
cult	4.27	3.10	3.45	3.40	3.16
drama	3.99	3.43	3.66	3.60	2.86
foreign	4.15	3.46	3.47	3.54	2.81
horror	3.90	3.38	3.52	3.47	2.91
independent	4.31	3.59	3.51	3.55	2.69
neo-noir	4.34	3.35	3.33	3.37	2.97
parody	4.13	3.36	3.35	3.28	2.73
romance	3.84	3.48	3.62	3.62	2.85
science fiction	3.99	3.55	3.33	3.57	2.73
tragedy	4.40	3.34	3.27	3.52	3.11
war	3.82	3.51	3.49	3.50	2.71

What we do next to make the algorithm more accurate is to take the active user's trait scores and the scores for each genre from the 5factor table and subtract them. Here follows an example on the Action and the Adventure genre

In the following table you can see the big five scoring for the active user and the scoring for the action and adventure genre.

Table 5.9: Calculating 50/50 1

USER PERSONALITY TRAITS

	ope	**con**	**ext**	**agr**	**neu**
active user	3,60	3.9	2.3	4,70	3,30

GENRES

	ope	**con**	**ext**	**agr**	**neu**
action	3,87	3.45	3.57	3,58	2,72
adventure	3,91	3.56	3.54	3,68	2,61

We now subtract each trait of the active user from the corresponding trait of all the different genres. We take the absolute value of the result because our main goal is to add all the final trait scores and calculate a distance metric. The higher the value of this distance metric for the user and a specific genre, the less he likes that genre. Here in our example with only the action and adventure genre in mind, we can say that our user has a slight preference towards the adventure genre as he has a lower score, meaning that his big five scores are closer to the scores of the adventure genre.

Table 5.10: Calculating 50/50 2

	ope	**con**	**ext**	**agr**	**neu**	**PERSONALITY SCORE**
action	|3.6-3.87|	|3.9-3.45|	|2.3-3.57|	|4.7-3.58|	|3.3-2.721	**3,69**
adventure	|3.6-3.91|	|3.9-3.56|	|2.3-3.54|	|4.7-3.68|	|3.3-2.61|	**3,6**

We do the same calculations for all the genres and then create a table that includes these results in an ascending order.

Table 5.11: Final 50/50 genre preferences

sci-fi	3,47
romance	3,51
war	3,59
adventure	3,6
horror	3,66
comedy	3,67
action	3,69
animatior	3,71
cartoon	3,73
drama	3,76
film-noir	3,98

These are the genre preferences of the active user based on his personality.

Next step is to rearrange the *Knn Movie List* based on these genre preferences. We will explain this with an example.

Let's assume that the Knn algorithm predicted that the active user will provide a rating of 4 on Star Wars and a rating of again a 4 on Psycho. Star wars is a sci-fi movie, and Psycho is a Horror movie. What our algorithm does, is to go and find which genre is each movie on the knn list and then make some calculations and provide the 50/50 and 80/20 lists.

So for the 50/50 way, our recommender takes the predicted rating of the Star Wars movie and divides it by 2. Then, assuming that Star Wars is only a Sci-Fi movie, it also takes the value of the sci-fi genre from the previous table and divides it with 2.

What this does, is that by diving by 2, we imply a 50 % to the Knn rating and a 50 % to the personality. If we add those two values, we get a new pseudo rating for the Star Wars movie, which is 3.735.

By applying the same concept to the Psycho movie, we can see that the new pseudo rating for the psycho movie is 3.83. These pseudo ratings are now nothing more than a distance from zero and again, the higher the value, the less you will like this movie.

So, if we only had the Knn suggestions, Psycho and Star Wars would be equally presented to the user as they both have the same predicted rating. However, with the addition of the personality factor, Star wars will be suggested first and then Psycho

Same thing with the 80/20 recommender, with the only difference that we are not dividing each value by 2, but we multiply the Knn predicted rating with 0.2 and the Personality genre rating with 0.8. That way, we put more emphasis on the personality than the Knn, because now personality is 80% and Knn 20%

If a movie belongs in more than one genre, we do the following:

Let's assume that the Knn suggests GoldenEye with a predicted rating of 4, which is both an Action and an Adventure movie. First, again we divide the Knn score by 2. Then, based on Table 5.3: Five Factor Table, we add 3.6 (adventure preference value) and 3.69 (action preference value) it by the number of genres this movies belongs to.

So for the 50/50 recommender we have:

$$\left(\frac{3.6+3.69}{2}\right)/2+\frac{4}{2}=3.822$$

Equation 5.3: 50/50 example equation

which is the final predicted rating for Golden Eye.

So, the last step is to recalculate the new distance values for each movie in the knn list and make it in ascending order, as now, the closer to 0 the better the recommendation. We do this for both the 50/50 and the 80/20 methods and print the three different lists to the user. Here follows an example:

Here are the genre preferences of the user:

Table 5.12: 50/50 User genre Preferences

cartoon	**2,38**
sci-fi	**2,42**
drama	**2,43**
horror	**2,43**
action	**2,45**
adventure	**2,52**
comedy	**2,57**
animatior	**2,59**
romance	**2,64**
war	**2,74**
film-noir	**3.02**

Here is the Knn movie list:

```
KNN recomendations:
Usual Suspects, The (1995)
Blade Runner (1982)
Silence of the Lambs, The (1991)
Empire Strikes Back, The (1980)
Amadeus (1984)
Schindler's List (1993)
One Flew Over the Cuckoo's Nest (1975)
Casablanca (1942)
It's a Wonderful Life (1946)
Rear Window (1954)
```

The 50/50 movie list:

```
50/50 recomendations:
Blade Runner (1982)
Silence of the Lambs, The (1991)
Empire Strikes Back, The (1980)
Alien (1979)
Psycho (1960)
Amadeus (1984)
Schindler's List (1993)
One Flew Over the Cuckoo's Nest (1975)
Casablanca (1942)
It's a Wonderful Life (1946)
```

The 80/20 movie list

```
80/20 recomendations:
Lion King, The (1994)
Blade Runner (1982)
Silence of the Lambs, The (1991)
Wallace & Gromit: The Best of Aardman Animation
Wrong Trousers, The (1993)
L.A. Confidential (1997)
Close Shave, A (1995)
Maltese Falcon, The (1941)
Beauty and the Beast (1991)
Faust (1994)
```

We can clearly see that the 80/20 list is really close to the personality genre preferences of the user, but the most well balanced is the 50/50 one, as we also found out during our evaluation of the recommender.

On the 50/50 list, we see the Sci-fi movies rising in the rankings and even horror movies like Alien and Psycho appeared on the list, where previously on the Knn list they weren't included. This is because the user has a preference on Sci-fi and Horror movies based on Table 5.12: 50/50 User genre Preferences.

On the 80/20 list, we see that out of nowhere, a cartoon movie is suggested first, because cartoon is the favorite genre of the active user. We also see the appearance of more cartoon movies like Beauty and the Beast, and Wallace & Gromit: The Best of Aardman Animation.

6 IMPLEMENTATION IN PYTHON

6.1 Functions

exit_if_file_is_not_found(filename) :

This is a function that checks if a file being accessed exists, and closes the program if it doesn't. It is used each time a file is loaded.

exit_if_data_are_not_found() :

Calls the exit_if_file_is_not_found function for all needed files.

get_user_movie_id() :

This function gets, checks and validates the user input for the IDs of movies he rates. While the input is not valid, the function asks for new input in a loop. If the input is valid, it returns it.

get_user_rating(TAINIA) :

This function gets, checks and validates the user input for the rating of movies he rates. While the input is not valid, the function asks for new input in a loop. If the input is valid, it returns it.

newUser(ratingsmatrix) :

This function creates a new user and prompts him/her to rate 20 movies. It reads movies from a file that contains the movies sorted by popularity (number of votes) and then presents to the user a list of 20 movies each time: 10 movies from the top of the list and 10 random choices. The user can pick a movie by ID and rate it (using get_user_movie_id and get_user_rating functions) or refresh the list.

myBestMovies(me,N) :

This function retrieves the N best rated movies for a given user.

correlation(u,v) :

Finds the Pearson correlation between two users.

userPairSimilarity(user1, user2) :

Returns the similarity between two users, by checking their common movies and using the above correlation function.

nearestNeighbourPredictions(user, K) :

This function calls the userPairSimilarity function for the active user and all other users. Then it uses the K most similar users and their ratings to predict a rating that the active user would give for all the movies in our dataset. It returns an array that contains all movies and their predicted rating.

finalNRecommendations(user, N) :

This function takes the predicted ratings array calling the above function, sorts it in descending order (best rating to worst) and returns the N top ones, after droping all movies already rated by the active user.

RatingsNormalized(user) :

This function normalizes the ratings in the predicted ratings array in the range (0-5).

knnPersonality5050(genreSum, myItemDatabase, movieNum, genreNum, normalizedTable):

and

knnPersonality8020(genreSum, myItemDatabase, movieNum, genreNum, normalizedTable):

These functions take the normalized KNN ratings from RatingsNormalized and alter them based on the personality of the user, equally weighting KNN and personality in the first case and favoring the KNN in the second case. They return a new normalized table with revised ratings.

finalNRecommendationsPersonality(tab, user, N) :

Takes the new personality-based normalized rating tables and returns the N top movies.

6.2 Program Flow, Details And Explanation

Lines 8-9:

At first, the program gets its path name and directory to use for locating all the neccessary files regardless of the machine it's being run on.

Lines 12-16:

The exit_if_file_is_not_found function is defined.

Lines 18-26:

The program then checks on what OS the program is run (by checking the os.name attribute) and sets a descriptor variable to "/" in the case of Posix, or "\\" otherwise, to ensure that the program will run on different platforms:

```
if os.name == 'posix':
    descriptor = '/'
    def clear_screen():
        os.system('clear')
else:
    descriptor = '\\'
    def clear_screen():
        os.system('cls')
```

Lines 28-32:

Here the paths to the data needed by the program are defined, concatinating the directory string retrieved before (line 9), the descriptor variable ('\\' or '/' depending on the OS) and the file name. The files defined are:

movieLensFile: The u.data file from the movieLens database, containing the ratings of all users for all movies they have rated.

movieLensItemFile: The u.item file from the movieLens database, containing information about the movies such as name, actors and genres.

FiveFactorFile:

mostRatedFile: A file containing all movies sorted by the number of votes (most to least).

Lines 34-39:

Next, the program checks if the files defined above exist by calling the exit_if_file_is_not_found function for each file. If a file does not exist, the function aborts the program.

Lines 42-73:

If all needed files exist, the program defines the get_user_movie_id function. This function is used inside the newUser function that gets defined later, and gets/checks the user input for the movie id he wants to rate, inside a loop. If the input is valid the loop breaks and the function returns it, but it keeps asking for new input if:

a)The input has a '0' prefix (leads to failure if left unchecked)

```
if choice == '01' or \
   choice == '02' or \
   choice == '03' or \
   choice == '04' or \
   choice == '05' or \
   choice == '06' or \
   choice == '07' or \
   choice == '08' or \
   choice == '09':
    print 'Wrong Movie ID.
```

b)The input is outside the [0,1682] range (valid movie IDs) and is not -1 (valid input - it is used to refresh the list)

```
if choice < -1 or choice > 1682 or choice == 0:
    # Not acceptable integer value
    print 'Wrong Movie ID. Please try again ...'
    continue
```

c)The input is not an integer

```
try:
    choice = int(choice) # Try to convert string to integer
except ValueError:
    # If it fails, ask the user again
    print "Wrong Movie ID. This is not a number. Please try again ..."
    continue
```

Lines 75-107:

In the next lines, the function get_user_rating is defined. It asks the user to rate the selected movie and takes the name of the movie as an argument (in order to print it when asking for a rating). Like the previous function, it asks for input in a loop, which breaks only if the input is valid, otherwise it keeps asking. The input is considered not valid if:

a)The input has a '0' prefix (similar case to the one in the getUserRating function)

b)The input is not an integer:

```
try:
    choice = int(choice) # Try to convert string to integer
except ValueError:
    # If it fails, ask the user again
    print "Wrong Rating. This is not a number. Please try again ..."
    continue
```

c)The input is outside the [1,5] range (valid rating values):

```
# If the conversion to integer has been succedded, check for its value
if choice < 1 or choice > 5:
    # Not acceptable integer value
    print 'Wrong Rating. Please insert a value between [1-5]:'
    continue
```

If the input is valid, it is returned.

Lines 109-255:

Next, the function newUser is defined. This function initiates a loop which repeatedly asks the user to rate movies from a list, until the user has rated 20 movies. The user can choose a movie from the list (by ID) to rate, or refresh the list by giving -1 as input.

After informing the user on what he needs to do, the function loads a pandas DataFrame (myRatedDatabese) containing all the movies available in descending order (most rated to least rated). It then defines a variable to be used as a counter for the number of movies rated (cnt) and enters a loop that is going to be terminated only when the counter is equal to 20. The movies are presented in the following fashion:

-Five integer variables are created, ii (used to mark the start of the section of the myRatedDatabase movies that the program is printing at any moment), jj (used to mark the end of it), icount (used to represent the row of the 20-row dataframe that the movie is going to be put in), jcount (used to represent the column of the dataframe we are handling at any moment - there are 2 columns, one for the movie ID and one for the movie name) and cnt (used to count the number of rated movies). jj is always equal to ii+10.

-A new dataframe, named movList, with space for 20 elements and two columns (MOVIE_NAME and MOVIE_ID) is created, to hold the 20 movies that will appear on screen each time.

-The program enters a loop that ends only when cnt is larger than 20 (the user has rated 20 movies).

-In every iteration, the program puts movies in the [ii,jj] range of the myRatedDatabase dataframe in the even numbered elements of the movList dataframe (1,3,5...17,19).

-For the odd numbered elements of movList (2,4,6....18,20), random movies from the [ii,1682] range are used:

```
for i in range(ii, jj):

    # row[1,3,5,7,9] are **not** random
    icount += 1
    movlist.loc[icount].loc['MOVIE_NAME'] = myRatedDatabase.iloc[i, 0]
    jcount += 1
    movlist.loc[icount].loc['MOVIE_ID'] = myRatedDatabase.iloc[i, 1]
    jcount -= 1

    # row[2,4,6,8,10] are random
    icount += 1
    ranc = randint(jj, 1682)
    movlist.loc[icount].loc['MOVIE_NAME'] = myRatedDatabase.iloc[ranc, 0]
    jcount += 1
    movlist.loc[icount].loc['MOVIE_ID'] = myRatedDatabase.iloc[ranc, 1]
    jcount -= 1
```

-When the movLIst is full, it is prented on screen and the user is asked to either select a movie id to rate or type -1 to refresh the list.

-User input is taken with the use of the get_user_movie_id function. Then, the program searched in movList for the selected movie id, and if it is not found it searches in the complete list of movies. If the movie id exists, the program asks the user to rate it with the use of the get_user_rating function.

Lines 229-234:

The myBestMovies function is defined. It simply sorts the movies rated by the user from highest to lowest rating and returns them.

Lines 238-244:

The correlation function, identical to the scipy function with the same name, meassures the Pearson correlation between two elements (users).

Lines 247-260:

The userPairSimilarity function is defined. It takes two user vectors (vectors that contain all movies and the ratings of a user for the movies this user has rated, or NaN if he hasn't rated a movie) representing two users, and returns the similarity between them.

First, the two vectors are normalized using the mean rating for each vector.

```
user1 = np.array(user1) - np.nanmean(user1)
user2 = np.array(user2) - np.nanmean(user2)
```

Then it finds the common movies between the two users. If they have rated no common movies, the function returns 0; otherwise, it collects the rating of each user for each of the common movies and calls the correlation function to return the correlation between them:

```
commonMovies = [i for i in range(len(user1)) if user1[i] > 0 and user2[i] > 0]
# Gives us movies for which both users have non NaN ratings
if len(commonMovies) == 0:
    # if there are no common movies that both users have rated then it returns 0
    return 0
else:
    user1 = np.array([user1[i] for i in commonMovies])
    user2 = np.array([user2[i] for i in commonMovies])
    return correlation(user1, user2)
```

Lines 265-283:

Next is the definition of the nearestNeighbourPredictions function. It takes two arguments, the userID of the active user and integer K, representing the number of neighbours we want to take into account.

First, a dataframe named similarities with a single column (labeled 'similarity') is created, which the program fills with the similarity of the active user with every other user in the database, calling the userPairSimilarity function for the active user and a different other user each time:

```
similarities = pd.DataFrame(index=userVectorMatrix.index,
                            columns=['Similarity'])
for i in userVectorMatrix.index:
    similarities.loc[i] = userPairSimilarity(userVectorMatrix.loc[user], userVectorMatrix.loc[i])
```

When the similarities dataframe is finished, it is sorted in descending order and only the first K elements ar kept (the similarities of the K most similar to the active user users) and saved in a dataframe named nearestNeighbours.

```
similarities = pd.DataFrame.sort_values(similarities, ['Similarity'], ascending=[0])
nearestNeighbours = similarities[:K]
neighbourVectors = userVectorMatrix.loc[nearestNeighbours.index]
```

Lines 287-307:

In the next lines, the function finalRecomendations is defined. This is the function that will return the final results of the KNN predictions. It takes four arguements: The active user ID, the number N of the movies to recomend, the total number of movies (movieNum) and the total number of genres (genreNum).

-First, the function calls the nearestNeighbourPredictions function to aquire a dataframe named predictRating containing the predicted rating of the active user based on his/her nearest neighbours.

```
predictRating = nearestNeighbourPredictions(user, neighbours)
```

-Then it makes a list containing the IDs of the movies the active user has already rated, named moviesRated.

```
moviesRated = list(userVectorMatrix.loc[user]
                   .loc[userVectorMatrix.loc[user] > 0].index)
```

-Next it creates an empty list named noGenreList, and fills it with all the movies from the myItemDatabase dataframe that have no genre, by iterating through all the columns of each row in the myItemDatabase dataframe and counting the ones in the genre section (each genre for each movie has a value of one, if that movie belongs to that genre, or zero if it doesn't).

```
noGenreList = []
for i in range(0, movieNum):
    cnt = 0
    for j in range(0, genreNum):
        if myItemDatabase.loc[i][j + 2] == 1:
            cnt = cnt + 1
    if cnt == 0:
        noGenreList.append(predictRating.index.get_loc(i+1))
```

-The function then drops from the recomendation dataframe (predictRating) all movies contained in the moviesRated and noGenreList lists, meaning it removes all movies that have no genre or are already rated by the user.

```
predictRating = predictRating.drop(noGenreList)
predictRating = predictRating.drop(moviesRated)
```

-Then the predictRating dataframe is sorted in descending order (highest prediction to lowest) and only the N first elements are kept. Using the movie IDs of those N elements, their titles are retrieved from the myItemDatabase dataframe and those titles are returned as a list.

```
finalRecommendations = pd.DataFrame.sort_values(predictRating, ['Rating'], ascending=[0])[:N]
titles = (myItemDatabase.loc[myItemDatabase.itemId.isin(finalRecommendations.index)])
return list(titles.title)
```

Lines 310-327:

Next a function named RatingNormalized is defined. This function takes the result of the nearestNeighbourPredictions function and normalizes their values, getting all ratings in the [1-5] range. Because the lowest predicted rating can also take negative values, this is done in the following manner:

-First the function gets the lowest and highest ratings on the predictRating dataframe (containing all movies with their predicted ratings) and stores them into a 'maximum' and a 'minimum' variables.

```
predictRating = nearestNeighbourPredictions(user, neighbours)
#print ("5050 preds "), predictRating
maximum = predictRating.iloc[:, 0].dropna().max()
minimum = predictRating.iloc[:, 0].dropna().min()
```

-Next, a variable b is defined, which is equal to 0 if the minimum value is not negative, and (-minimum) if it is negative. If the minimum value is negative, the function also changes the minimum variable to 0 and the maximum variable to maximum+|minimum|.

```
b = 0
if minimum < 0:
    b = -minimum + 0
    maximum = maximum - minimum
    minimum = minimum - minimum
```

-This variable b is then added to each predicted rating in the predictRating dataframe, to bring all values above 0 (if they already were, b is equal to 0 so this has no effect). All ratings are then recalculated in the range [1,5] and then they are reversed so that the closer they are to zero the better the rating is considered.

```
for i in predictRating.index:
    predictRating.loc[i] = predictRating.loc[i] + b
    predictRating.loc[i] = predictRating.loc[i] * (5 / maximum)
    predictRating.loc[i] = 5 - predictRating.loc[i]
```

-The revised predictRating dataframe is finally returned.

In line 383, the definitions end and normal program flow is continued. The exit_if_data_are_not_found function is called, and if the files needed exist, the program goes on and loads them into dataframes:

```
myUserDatabase = pd.read_csv(movieLensFile, sep="\t", header=None, names=['userId', 'itemId', 'rating']
myUserDatabase = newUser(myUserDatabase)
myItemDatabase=pd.read_csv(movieLensItemFile,sep="|", header=None, names=['itemId','title', 'Action','
myUserDatabase=pd.merge(myUserDatabase,myItemDatabase,left_on='itemId',right_on="itemId")
userVectorMatrix=pd.pivot_table(myUserDatabase, values='rating', index=['userId'], columns=['itemId'])
```

An active userID is also declared as 944 (this is the userID used for any new user, since userIDs 1-943 already exist) and a neighbour's variable, representing the number of nearest neighbours, we want the algorithm to take into account as 50 (lines 396-397).

Lines 399-1108:

Next comes the personality test that will determine the personality profile of the new user. Five variables, each representing a Big-Five personality trait, are declared:

```
a = 0
c = 0
o = 0
e = 0
n = 0
```

After printing some basic instructions (lines 405-412), the program asks the user to evaluate 50 questions giving a rating in the range [1,5], with 1 meaning 'I completely dissagree' and 5 'I completely agree'. First, the question is printed on screen, and then the user input is handles in a loop, so that the user is repeatedly asked for input while the input is not valid:

```
q = 'I am the life of the party.'

while (True):
    try:
        num = int(raw_input(q + '\n'))
    except ValueError:
        print 'Wrong input, please try again'
        continue
    if (num<1 or num>5):
        print 'Value must be in the range 1-5: Try again.'
        continue
    break
e = e+num
```

Each question adds or subtracts the given user rating from one of the variables representing each of the personality traits (e for extraversion in the above example). This goes on until all 50 questions are rated by the user. Then using the proccesses explained in the theory section of this study, the final values for the five personality traits are deduced (lines 1086 - 1091):

```
agreeableness_final = (24 + a)/10.0
openness_final = (18 + o)/10.0
conscientiousness_final =(24 + c)/10.0
extraversion_final = (30 + e)/10.0
neuroticism_final = (48 + n)/10.0
```

Next, the program prompts the user to wait and enters its final stage.

First, the 'myFiveFactorDatabase' DataFrame is created by reading the file variable 'FiveFactorFile' defined earlier:

```
myFiveFactorDatabase=pd.read_csv(FiveFactorFile,sep=",",header=None,
                names=['MOVIE GENRE','OPE','CON','EXT','AGR','NEU'], usecols=[0,1,2,3,4,5])
```

It contains the values of the Big Five traits expected for each of the genres in the movies database (the mean trait value for each genre). It contains eleven rows (one for each genre) and five columns (one for each trait), excluding the first column which contains the genres names.

After the DataFrame is created, the values deduced for the current user for each trait are subtracted from the values in the respective columns (opennes_final from all rows in the openness column, agreeableness_final from all rows in the agreeableness column and so on), and right after that the absolute value of each trait is taken so that no negative values remain. That way, the closer the user is to a specific trait value, the closer to zero the result will be:

```
myFiveFactorDatabase.loc[:, 'OPE']-=openness_final
myFiveFactorDatabase.loc[:, 'OPE'] = myFiveFactorDatabase.loc[:, 'OPE'].abs()
myFiveFactorDatabase.loc[:, 'CON']-=conscientiousness_final
myFiveFactorDatabase.loc[:, 'CON'] = myFiveFactorDatabase.loc[:, 'CON'].abs()
myFiveFactorDatabase.loc[:, 'EXT']-=extraversion_final
myFiveFactorDatabase.loc[:, 'EXT'] = myFiveFactorDatabase.loc[:, 'EXT'].abs()
myFiveFactorDatabase.loc[:, 'AGR']-=agreeableness_final
myFiveFactorDatabase.loc[:, 'AGR'] = myFiveFactorDatabase.loc[:, 'AGR'].abs()
myFiveFactorDatabase.loc[:, 'NEU']-=neuroticism_final
myFiveFactorDatabase.loc[:, 'NEU'] = myFiveFactorDatabase.loc[:, 'NEU'].abs()
```

Next, the program sums all the values for each row (each row represents a genre) and makes a new DataFrame, named "genreSum" containing the genre names and that final value for each genre. It's now obvious that the closer to zero the result is, the higher the chances the current user likes the given genre, according to the theory.

```
numbah = myFiveFactorDatabase.sum(axis=1, numeric_only=True)
numbahNames = myFiveFactorDatabase.loc[:, 'MOVIE GENRE']
genreSum=pd.concat([numbahNames, numbah], axis=1)
```

Now the program calls the RatingsNormalized function and stores the result in the "NormalizedTable" DataFrame. Then it calls the knnPersonality5050 and the knnPersonality5050 functions, giving the "RatingsNormalized" and the "genreSum" DataFrames as arguements:

```
tab = knnPersonality5050(genreSum, myItemDatabase, 1682, 11, normalizedTable)
```

```
tab2 = knnPersonality8020(genreSum, myItemDatabase, 1682, 11, normalizedTable)
```

Finally, the program computes and prints the final results for all three methods:

```
#print 'Your highest rated movies are: \n', myBestMovies(userID, 10)
l5050 = finalNRecommendationsPersonality(tab, userID, 10)
print '50/50 recomendations:'
for i in l5050:
    print(i)
print '\n\n'
lknn = finalNRecommendations(userID, 10, 1682, 11) #knn
print 'KNN recomendations:'
for i in lknn:
    print(i)
print '\n\n'
l8020 = finalNRecommendationsPersonality(tab2, userID, 10) #80/20
print '80/20 recomendations:'
for i in l8020:
    print(i)
print '\n\nPress Enter to Exit . . .'
useless = (raw_input())
```

7 EVALUATION AND FUTURE WORK

7.1 Evaluation

For the evaluation part, we sent a modified version of the recommender system to 30 different people. Out of these 30 people, 10 were female and 20 were male, aged between 18 and 65.

The only difference between the modified version and the final version of the recommender is that in the end of the program, in the modified version, we didn't show which result is the Knn, the 50/50 and the 80/20. The users were presented with three different sets of movies, and they had to pick between those 3 in order of preference. The user had to pick his favourite set based on both what movies were included in the set, but also the order of the movies in each set.

The users replied via email, in the style, of " I firstly, Prefer set A, then set B and last set C." We then created an Excel sheet so to count the final score for each method. The first preference of each user was awarded 3 points, the second two points and the last one 1 point. You can see the results in the next table.

Table 7.1:50/50 Evaluation Table

gender	50 50	knn	80 20	age
f	3	2	1	50
m	2	1	3	26
f	2	3	1	23
m	1	3	2	23
m	2	3	2	29
m	3	2	1	33
m	2	3	1	40
m	3	2	3	30
m	2	3	1	20
f	3	1	2	30
m	3	2	3	60
f	3	1	2	24
f	1	3	2	28
f	2	2	3	18
f	3	2	1	21
m	3	2	2	19
f	2	3	3	30
m	3	3	2	34
m	3	3	2	34
m	3	1	2	32
m	3	3	2	28
m	2	3	1	23
f	3	1	2	30
m	2	3	1	30
m	2	3	1	40
m	1	2	3	35
m	3	1	2	25
m	2	1	3	50
m	3	2	1	30
f	2	3	1	36

There are some cases were, for example, the 50/50 recommender and the Knn presented exactly the same movies with the exact order. This is why you might see the same points awarded for the two different methods.

Of course, our main goal was to prove that most people prefer the 50/50 recommender over the Knn. The 80/20 was an exaggeration and was just provided as an option so that we have an algorithm that uses mostly the personality, as the 80/20 is 80 percent personality and 20 percent Knn.

The results can be seen on the following table:

Table 7.2: 50/50 Evaluation Table 2

	50/50	**knn**	**80/20**
SUM	72	67	56
%	36.41026	34.35897	29.23077

We can see that there is a small preference towards the 50/50 recommender but if we eliminate the 80/20 recommender and recalculate the results, we can notice an improvement of 3.62 % which is a significant difference.

Table 7.3: 50/50 Evaluation Table 3

	50/50	**knn**
SUM	72	67
%	52.17391	48.55072

7.2 Conclusions

The general conclusion of this Dissertation project is that indeed personality plays a significant part in recommender systems. We believe that as computer technology progresses, human personality and psychology must be further integrated and applied and with this project, we made a point on how important personalization can be in the modern market. Our 50/50 recommender and the percentage of improvement that it made to the Knn algorithm, even though it may seem as a small percentage, it's actually pretty significant, and if you translate that into market value and sales, it makes a big difference.

For some, answering a personality test might be a big effort and something not needed, but you have to make sure that they understand that this test will improve their experience and their recommendations. Personally, we believe that it's worth the effort, even more because it's something you have to do only once and the outcome will be significant even in the future.

7.3 Improvements And Future Work

A first improvement would be to include more metadata in the future. The only metadata we had now was what genre each movie was. Of course, this is something that the current MovieLens database purely provided, but we can scrap metadata from IMDB and incorporate more content based techniques.

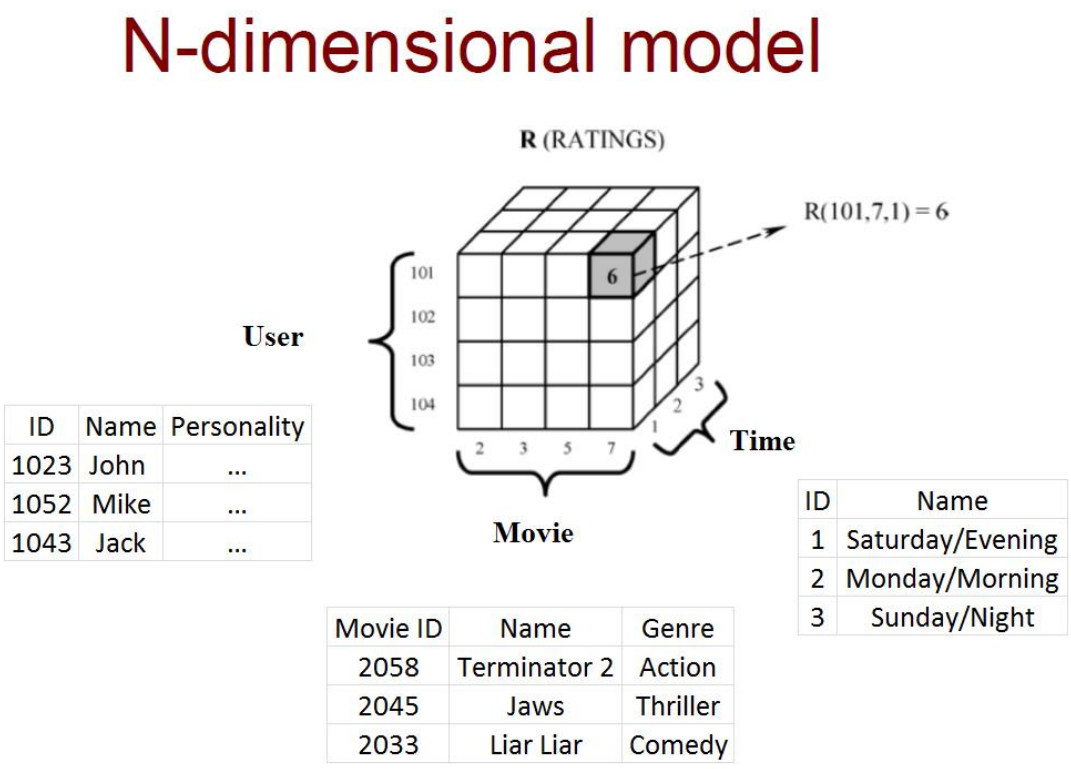
Furthermore, the movie database is again pretty old, and we need to include more modern movies. This was a common complain among young people who evaluated the recommender, as they haven't watched most of the movies included, even though they are relatively famous.

Another addition would be to be able to treat a new movie entering the system and not only a new user.

The MovieLens database includes also the gender, age and occupation of each user. In the future, all of this data can be incorporated so to cluster the users and be able to make more accurate predictions. Clustering should be tested even though in the past it didn't provide the best results.

Feedback from the user was something we definitely wanted to incorporate but didn't have the time to. User must provide feedback for the recommendations and rate the recommendations. Then we will have three important factors to worry about, his personality, his movie preferences and his reaction to the recommendations. Moreover, the user should be able to provide feedback for different aspects of the movie like acting, plot, movie length, director and so on. Nowadays, recommenders assume that if you like a movie, you like anything about the movie, the director, the actors, etc. So, if you provide specific feedback as stated above, the recommendations will be more accurate.

In the beginning of this project, we wanted to incorporate a group option to the recommender meaning that the recommender system would combine two or more different users so to make common recommendations. This group option would consider both user's personalities and movie preferences and find the optimal list of movies for them to watch. A further idea was also to incorporate metadata like place, date and time in conjunction with the companion option so that you can get a more specific recommendation. This will have to be done in an N-dimensional model like the one provided below.



ID	Name	Personality
1023	John	...
1052	Mike	...
1043	Jack	...

ID	Name
1	Saturday/Evening
2	Monday/Morning
3	Sunday/Night

Movie ID	Name	Genre
2058	Terminator 2	Action
2045	Jaws	Thriller
2033	Liar Liar	Comedy

Picture 7.1: N-dimensional model

Last but not least, a more advanced personality test will be used, even a custom one. We are currently looking at other options and trying to minimize the number of questions and effort the user has to make to portray his personality into our movie recommender.

BIBLIOGRAPHY AND REFERENCES

Bibliography

[1] Robin Burke, "Hybrid Recommender Systems:Survey and Experiments".

[2] Alexander Tuzhilin Gediminas Adomavicius, "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions," *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 17, NO. 6*, Jun 2005.

[3] John Riedl, Joseph A. Konstan Sean M. McNee, "Making Recommendations Better: An Analytic Model for Human-Recommender Interaction," *CHI*, April 2006.

[4] Ignacio Fernández-Tobías, Alejandro Bellogín Iván Cantador, "Relating Personality Types with User Preferences in Multiple Entertainment Domains".

[5] Suhaas Prasad. Using Social Networks to Improve Movie Rating Predictions.

[6] Andy Mai Mathangi Venkatesan, "Recommendation of TV shows and Movies based on Facebook data".

[7] Mohammad-Hossein Nadimi-Shahraki and Mozhde Bahadorpour, "Cold-start Problem in Collaborative Recommender Systems: Efficient Methods Based on Ask-to-rate Technique".

[8] Istvan Albert, Dan Cosley, Shyong K. Lam, Sean M. McNee, Joseph A. Konstan, John Riedl Al Mamunur Rashid, "Getting to Know You: Learning New User Preferences in Recommender Systems".

[9] George Karypis, and John Riedl Al Mamunur Rashid, "Learning Preferences of New Users in Recommender Systems: An Information Theoretic Approach".

[10] Fan Min, Xu He, Yuan-Yuan Xu1 Heng-Ru Zhang, "A Hybrid Recommender System Based on User-Recommender Interaction," *Hindawi Publishing Corporation Mathematical Problems in Engineering*, p. Article ID 145636, 2015.

[11] Saranya Maneeroj Nutcha Rattanajitbanjong, "Multi criteria pseudo rating and multidimensional user profile for movie recommender system," in *2nd IEEE International Conference*, 2009.

[12] Eric Norris Jennifer Golbeck, "Personality, movie preferences and recommendations," in *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2013.

[13] Vladimir-Nicolae Dinu, Catalina Preda, Matei Macri Costin-Gabriel Chiru, "Movie Recommender System Using the User's Psychological Profile," *IEEE*, pp. 978-1-4673-8200-7/15, 2015.

[14] personality-testing.info, "The Big Five Personality Test," ipip.ori.org,.

[15] Alexander Felfernig and Mehmet H. Göker Robin Burke, "Recommender Systems: An Overviw," *AI MAGAZINE*, Fall 2015.

[16] Barbara Cimpa Jochen Nessel, "The MovieOracle - Content Based Movie Recommendations," *ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, 2011.

[17] Matus Medob, Chi Ho Yeungb, Yi-Cheng Zhanga, Zi-Ke Zhanga Linyuan Lua, "Recommender Systems," *physics.soc-ph*, Feb 2012.

[18] Arkadiusz Paterek, "Predicting movie ratings and recommender systems," Jun 2012.

[19] Ghadeer Shahatah, Lamia Albdulkarim Dhoha Almazro, "A Survey Paper on Recommender Systems.," *cs.IR*, Dec 2010.

[20] Stephan Spiegel, A Hybrid Approach to Recommender Systems.

[21] Xiangrui Meng, Chao Liu, Qiang Yang Nathan Liu. Wisdom of the Better Few: Cold Start Recommendation via Representative based Rating Elicitation.

[22] Gaurangi Tilak, Nan Li Eyrun A. Eyjolfsdottir, "MovieGEN: A Movie Recommendation System," Santa Barbara,.

[23] Emrah Ekmekciler Hasan Ogul, Two-Way Collaborative Filtering on Semantically Enhanced Movie Ratings, Jun 25-28, 2012.

[24] A. M. Jehad Sarkar, Young-Koo Lee Sajal Haider, Movie Recommendation

System Based on Movie Swarm, 2012.

[25] Sandeep Matharia, Dr. C.N.S. Mmthy Dharmendra Patliak, "ORBIT: HYBRID MOVIE RECOMMENDATION ENGINE," *International Conference on Emerging Trends in Computing, Communication and Nanotechnology*, 2013.

[26] Maria Augusta Silveira Netto Nunes. (2008, Dec) Recommender Systems based on Personality Traits. [Online]. https://tel.archives-ouvertes.fr/tel-00348370

[27] M. KUNAVER, M. POGACNIK, A. KOSIR AND J. F. TASIC Pozrl T., "IMPROVING HUMAN-COMPUTER INTERACTION IN PERSONALIZED TV RECOMMENDER," *Transactions of Electrical Engineering*, pp. 19-36, Jun 2012.

[28] Joseph A. Konstan Maxwell F. Harper, "The MovieLens Datasets: History and Context," *ACM Digital Library*, Dec 2015.

Pictures

Equations

Tables

Script

```
1    import sys, os
2    import numpy as np
3    import pandas as pd
4    from numpy import linalg as LA
5    from random import randint
6    from scipy.sparse.csgraph import _validation
7
8    pathname = os.path.dirname(sys.argv[0])
9    directory = os.path.abspath(pathname)
10
11
12   # If you cannot find the given 'filename', then exit (1)
13   def exit_if_file_is_not_found(filename):
14       if not os.path.exists(filename):
15                       print("Required  data  file  is  missing:  {0}
".format(filename))
16           sys.exit(1)
17
18   # Make it cross-platform (Windows + Linux)
19   if os.name == 'posix':
20       descriptor = '/'
21       def clear_screen():
22           os.system('clear')
23   else:
24       descriptor = '\\'
25       def clear_screen():
26           os.system('cls')
27
28   # Define the required data
29   movieLensFile = directory + descriptor + 'u.data'
30   movieLensItemFile = directory + descriptor + 'u.item'
31   FiveFactorFile= directory + descriptor + '5factortable.csv'
32   mostRatedFile = directory + descriptor + 'mostratedmovies.csv'
33
34   # Test requirements
35   def exit_if_data_are_not_found():
36       exit_if_file_is_not_found(movieLensFile)
37       exit_if_file_is_not_found(FiveFactorFile)
38       exit_if_file_is_not_found(movieLensItemFile)
39       exit_if_file_is_not_found(movieLensFile)
40
41
42   def get_user_movie_id():
43       q = 'Put the movie ID of the movie you want to rate, or -1 to
refresh list:'
44       # Fix the '0{1,2,3,4,5,6,7,8,9}' bug
45       while True:
46           choice = raw_input(q + '\n')
47           if choice == '01' or \
48              choice == '02' or \
49              choice == '03' or \
50              choice == '04' or \
51              choice == '05' or \
52              choice == '06' or \
53              choice == '07' or \
54              choice == '08' or \
55              choice == '09':
56                  print 'Wrong Movie ID. Please do not use 0 prefix!
Try again ...'
```

```
57          else:
58              # We are not affected from the bug
59              try:
60                    choice = int(choice) # Try to convert string to
integer
61              except ValueError:
62                  # If it fails, ask the user again
63                      print "Wrong Movie ID. This is not a number.
Please try again ..."
64                  continue
65
66                # If the conversion to integer has been succedded,
check for its value
67              if choice < -1 or choice > 1682 or choice == 0:
68                  # Not acceptable integer value
69                  print 'Wrong Movie ID. Please try again ...'
70                  continue
71              # It's an acceptable integer value
72              break # Break the loop
73      return choice
74
75  def get_user_rating(TAINIA):
76      movie = str(TAINIA)
77      q = "How much would you rate '" + movie + "' ?"
78      # Fix the '0{1,2,3,4,5,6,7,8,9}' bug
79      while True:
80          choice = raw_input(q + '\n')
81          if choice == '01' or \
82             choice == '02' or \
83             choice == '03' or \
84             choice == '04' or \
85             choice == '05' or \
86             choice == '06' or \
87             choice == '07' or \
88             choice == '08' or \
89             choice == '09':
90               print 'Wrong Rating. Please do not use 0 prefix. Try
again ...'
91          else:
92              # We are not affected from the bug
93              try:
94                    choice = int(choice) # Try to convert string to
integer
95              except ValueError:
96                  # If it fails, ask the user again
97                   print "Wrong Rating. This is not a number. Please
try again ..."
98                  continue
99
100               # If the conversion to integer has been succedded,
check for its value
101             if choice < 1 or choice > 5:
102                 # Not acceptable integer value
103                       print 'Wrong Rating. Please insert a value
between [1-5]:'
104                 continue
105             # It's an acceptable integer value
106             break # Break the loop
107     return choice
108
109 def newUser(ratingsmatrix):
```

```
110       userID = 944
111       print 'Welcome!\n'
112         print 'We will present you a list of movies for rating.
\nPlease enter the ID of the movie you want to rate and press
Enter.\nThen put the corresponding rating (integer value in the range
1-5), with 1 being bad and 5 being excellent'
113       print '\n\nPress Enter to continue . . .'
114       useless = (raw_input())
115       print '\n\n'
116            myRatedDatabase = pd.read_csv(mostRatedFile, sep=",",
header=None, names=['MOVIE_NAME', 'MOVIE_ID'], usecols=[0, 1])
117       print '\n\nMOVIE LIST\n\n'
118       ii = -9
119       jj = 1
120       cnt = 1
121                movlist = pd.DataFrame(index=range(0, 21, 1),
columns=['MOVIE_NAME', 'MOVIE_ID'])
122       while cnt <= 20:
123           clear_screen()
124           print "MOVIE = " + str(cnt) + "/ 20"
125            print "-----------------------------------------------
---"
126           icount = 0
127           jcount = 0
128           ii += 10
129           jj += 10
130           movlist.loc[0].loc['MOVIE_NAME'] = ' '
131           movlist.loc[0].loc['MOVIE_ID'] = ' '
132
133            # Refresh the movie list with [ 21 x 2 ] -- print begins
from row[1] and ends at row [20]
134            # ------------------------------------------------------
-----
135           for i in range(ii, jj):
136
137               # row[1,3,5,7,9] are **not** random
138               icount += 1
139                                movlist.loc[icount].loc['MOVIE_NAME'] =
myRatedDatabase.iloc[i, 0]
140               jcount += 1
141                                 movlist.loc[icount].loc['MOVIE_ID'] =
myRatedDatabase.iloc[i, 1]
142               jcount -= 1
143
144               # row[2,4,6,8,10] are random
145               icount += 1
146               ranc = randint(jj, 1682)
147                                movlist.loc[icount].loc['MOVIE_NAME'] =
myRatedDatabase.iloc[ranc, 0]
148               jcount += 1
149                                 movlist.loc[icount].loc['MOVIE_ID'] =
myRatedDatabase.iloc[ranc, 1]
150               jcount -= 1
151
152           # Print to screen
153                    print '\t\t', movlist.to_string(index = False,
header=False)
154
155
156
```

```
157             # User input: Ask the user either to pick a Movie or
refresh the list
158             # ------------------------------------------------------
----
159             choice = get_user_movie_id()
160             if choice == -1:
161                 # Redesign the movielist dataframe (+10 new movies)
162                 continue
163
164
165
166             # Find the name of the movie based on the User input
167             # ------------------------------------------------------
-----
168             found = False
169             for row in enumerate(movlist.values):
170                 MOVIE_INDEX = row[0]
171                 MOVIE_NAME = row[1][0]
172                 MOVIE_ID = row[1][1]
173                 if ( MOVIE_INDEX == 0 ):
174                     # The first row is always empty, please SKIP it
175                     continue
176                 if choice == int(MOVIE_ID):
177                     TAINIA = MOVIE_NAME
178                     found = True
179                     break
180                 elif MOVIE_INDEX >= 1 and MOVIE_INDEX <=20:
181                     # Search withing the 20 Listed movies atm
182                     continue
183
184             # User selected a number that's not currently displays on
top 20 list (cheat)
185             if not found:
186                 for row2 in enumerate(myRatedDatabase.values):
187                     MOVIE_INDEX = row2[0]
188                     MOVIE_NAME  = row2[1][0]
189                     MOVIE_ID    = row2[1][1]
190                     if ( MOVIE_INDEX == 0 ):
191                         continue
192                     if choice == int(MOVIE_ID):
193                         TAINIA = MOVIE_NAME
194                         found = True
195                         break
196                     else:
197                         continue
198
199             if not found:
200                   print "Internal error. I cannot find the movie with
such ID."
201             exit(2)
202
203
204
205             # User input: Ask the to rate the selected movie
206             # ------------------------------------------------------
207
208             rate = get_user_rating(TAINIA)
209             print 'You rated ', TAINIA, ' with ', rate
210
211
212
```

```
213              # Record the user's opinion into the database (aka
ratingmatrix)
214           # ------------------------------------------------------
--
215                   df  =  pd.DataFrame([[long(userID),  long(choice),
long(rate)]], columns=('userId', 'itemId', 'rating'))
216           ratingsmatrix.loc[len(ratingsmatrix)] = df.loc[0]
217
218
219
220           # Keep voting until 20 Movies
221            # ------------------------------------------------------
---
222           cnt += 1
223
224       print '\n\nPlease Wait . . .\n\n'
225       return ratingsmatrix
226
227
228
229   def myBestMovies(me,N):
230
231       topNMovies=pd.DataFrame.sort_values(
232
myUserDatabase[myUserDatabase.userId==me],['rating'],ascending=[0])[:N
]
233
234       return list(topNMovies.title)
235
236
237
238   def correlation(u,v):
239       umu = u.mean()
240       vmu = v.mean()
241       um = u - umu
242       vm = v - vmu
243       dist = 1.0 - np.dot(um, vm) / (LA.norm(um) * LA.norm(vm))
244       return dist
245
246
247   def userPairSimilarity(user1, user2):
248       user1 = np.array(user1) - np.nanmean(user1)
249       user2 = np.array(user2) - np.nanmean(user2)
250
251
252        commonMovies = [i for i in range(len(user1)) if user1[i] > 0
and user2[i] > 0]
253       # Gives us movies for which both users have non NaN ratings
254       if len(commonMovies) == 0:
255              # if there are no common movies that both users have
rated then it returns 0
256           return 0
257       else:
258           user1 = np.array([user1[i] for i in commonMovies])
259           user2 = np.array([user2[i] for i in commonMovies])
260           return correlation(user1, user2)
261
262
263
264
265   def nearestNeighbourPredictions(user, K):
```

```
266
267       similarities = pd.DataFrame(index=userVectorMatrix.index,
268                                   columns=['Similarity'])
269       for i in userVectorMatrix.index:
270                                               similarities.loc[i]     =
userPairSimilarity(userVectorMatrix.loc[user],
userVectorMatrix.loc[i])
271             similarities  =  pd.DataFrame.sort_values(similarities,
['Similarity'], ascending=[0])
272       nearestNeighbours = similarities[:K]
273                                               neighbourVectors        =
userVectorMatrix.loc[nearestNeighbours.index]
274        predictRating = pd.DataFrame(index=userVectorMatrix.columns,
columns=['Rating'])
275       for i in userVectorMatrix.columns:
276           prediction = np.nanmean(userVectorMatrix.loc[user])
277           for j in neighbourVectors.index:
278               # for each neighbour in the neighbour list
279               if userVectorMatrix.loc[j, i] > 0:
280                   prediction += (userVectorMatrix.loc[j, i]
281                                                                          -
np.nanmean(userVectorMatrix.loc[j]))     *     nearestNeighbours.loc[j,
'Similarity']
282           predictRating.loc[i, 'Rating'] = prediction
283       return predictRating
284
285
286
287   def finalNRecommendations(user, N, movieNum, genreNum):
288
289       predictRating = nearestNeighbourPredictions(user, neighbours)
290       #print 'KNN predictions ', predictRating
291       moviesRated = list(userVectorMatrix.loc[user]
292                                     .loc[userVectorMatrix.loc[user] >
0].index)
293       noGenreList = []
294       for i in range(0, movieNum):
295           cnt = 0
296           for j in range(0, genreNum):
297               if myItemDatabase.loc[i][j + 2] == 1:
298                   cnt = cnt + 1
299           if cnt == 0:
300               noGenreList.append(predictRating.index.get_loc(i+1))
301
302        noGenreList = set(noGenreList) - set(moviesRated) #remove the
movies that exist inside the moviesRated list
303       predictRating = predictRating.drop(noGenreList)
304       predictRating = predictRating.drop(moviesRated)
305                                           finalRecommendations       =
pd.DataFrame.sort_values(predictRating, ['Rating'], ascending=[0])[:N]
306                                                    titles        =
(myItemDatabase.loc[myItemDatabase.itemId.isin(finalRecommendations.in
dex)])
307       return list(titles.title)
308
309
310   def RatingsNormalized(user):
311
312       predictRating = nearestNeighbourPredictions(user, neighbours)
313       #print ("5050 preds "), predictRating
314       maximum = predictRating.iloc[:, 0].dropna().max()
```

```
315       minimum = predictRating.iloc[:, 0].dropna().min()
316
317       b = 0
318       if minimum < 0:
319           b = -minimum + 0
320           maximum = maximum - minimum
321           minimum = minimum - minimum
322
323       for i in predictRating.index:
324           predictRating.loc[i] = predictRating.loc[i] + b
325                predictRating.loc[i] = predictRating.loc[i] * (5 /
maximum)
326           predictRating.loc[i] = 5 - predictRating.loc[i]
327       return predictRating
328
329      def  knnPersonality5050(genreSum,  myItemDatabase,  movieNum,
genreNum, normalizedTable):
330       normalizedTable2 = normalizedTable.copy(deep=1)
331       for i in range(0, movieNum):
332           cnt = 0
333           sum = 0
334           for j in range(0, genreNum):
335                       sum = sum + myItemDatabase.loc[i][j + 2] *
genreSum.loc[j, 0]
336               if myItemDatabase.loc[i][j + 2] == 1:
337                   cnt = cnt + 1
338           if cnt != 0:
339               sum = sum / cnt
340                normalizedTable2.loc[i + 1] = (normalizedTable.loc[i
+ 1] + sum) / 2
341           else:
342               normalizedTable2.loc[i + 1] = 5
343
344       return normalizedTable2
345
346      def  knnPersonality8020(genreSum,  myItemDatabase,  movieNum,
genreNum, normalizedTable):
347       normalizedTable3 = normalizedTable.copy(deep=1)
348       for i in range(0, movieNum):
349           cnt = 0
350           sum = 0
351           for j in range(0, genreNum):
352                       sum = sum + myItemDatabase.loc[i][j + 2] *
genreSum.loc[j, 0]
353               if myItemDatabase.loc[i][j + 2] == 1:
354                   cnt = cnt + 1
355           if cnt != 0:
356               sum = sum / cnt
357                normalizedTable3.loc[i + 1] = (normalizedTable.loc[i
+ 1] * 0.2) + (sum * 0.8)
358           else:
359               normalizedTable3.loc[i + 1] = 5
360
361       return normalizedTable3
362
363
364   def finalNRecommendationsPersonality(tab, user, N):
365       moviesRated = list(userVectorMatrix.loc[user]
366                                   .loc[userVectorMatrix.loc[user] >
0].index)
367
```

```
368
369       tabRated = tab.drop(moviesRated)
370
371       finalRecommendations = pd.DataFrame.sort_values(tabRated,
372                                                       ['Rating'],
ascending=[1])[:N]
373
374                                                       titles        =
(myItemDatabase.loc[myItemDatabase.itemId.isin(finalRecommendations.in
dex)])
375       return list(titles.title)
376
377
378
379
#########################################################################
###
380   # Basic tests #  If one of the following tests fails, then exit
the programm immediatelly  #
381
#########################################################################
####
382
383   exit_if_data_are_not_found()
384
385
386
387   ############
388   ### Main ###
389   ############
390
391         myUserDatabase    =    pd.read_csv(movieLensFile,    sep="\t",
header=None, names=['userId', 'itemId', 'rating'], usecols=[0, 1, 2])
392   myUserDatabase = newUser(myUserDatabase)
393                 myItemDatabase=pd.read_csv(movieLensItemFile,sep="|",
header=None,     names=['itemId','title',     'Action','     Adventure',
'Animation',   'Cartoon',   'Comedy','Drama',   'Film-Noir',   'Horror',
'Romance',                'Sci-Fi',                'War'                ],
usecols=[0,1,6,7,8,9,10,13,15,16,19,20,22])
394
myUserDatabase=pd.merge(myUserDatabase,myItemDatabase,left_on='itemId'
,right_on="itemId")
395    userVectorMatrix=pd.pivot_table(myUserDatabase,  values='rating',
index=['userId'], columns=['itemId'])
396   userID = 944;
397   neighbours=50
398
399   a = 0
400   c = 0
401   o = 0
402   e = 0
403   n = 0
404   while(True):
405       print '\nWELCOME TO THE BIG FIVE PERSONALITY TEST\n\n'
406       s='This is a personality test to help us understand  how your
personality is structured and find the best movie recommendations for
you.'
407       l='Please answer all the following questions with a number in
the  range  of  1-5,  where  1=disagree,  2=slightly  disagree,  3=neutral,
4=slightly agree and 5=agree.'
408       print s
```

```
409       print l
410
411       print '\n\nAnswer the following questions, with the prefix "I
think that.."\n\n\n'
412       q = 'I am the life of the party.'
413
414       while (True):
415           try:
416               num = int(raw_input(q + '\n'))
417           except ValueError:
418               print 'Wrong input, please try again'
419               continue
420           if (num<1 or num>5):
421               print 'Value must be in the range 1-5: Try again.'
422               continue
423           break
424       e = e+num
425
426
427       q = 'I feel little concern for others.'
428       while (True):
429           try:
430               num = int(raw_input(q + '\n'))
431           except ValueError:
432               print 'Wrong input, please try again'
433               continue
434           if (num<1 or num>5):
435               print 'Value must be in the range 1-5: Try again.'
436               continue
437           break
438       a = a-num
439
440       q = 'I am always prepared.'
441       while (True):
442           try:
443               num = int(raw_input(q + '\n'))
444           except ValueError:
445               print 'Wrong input, please try again'
446               continue
447           if (num<1 or num>5):
448               print 'Value must be in the range 1-5: Try again.'
449               continue
450           break
451       c = c+num
452
453       q = 'I get stressed out easily.'
454       while (True):
455           try:
456               num = int(raw_input(q + '\n'))
457           except ValueError:
458               print 'Wrong input, please try again'
459               continue
460           if (num<1 or num>5):
461               print 'Value must be in the range 1-5: Try again.'
462               continue
463           break
464       n = n-num
465
466       q = 'I have a rich vocabulary.'
467       while (True):
468           try:
```

```
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    o = o+num

    q = 'I do not talk a lot.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    e = e-num

    q = 'I am interested in people.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    a = a+num

    q = 'I leave my belongings around.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    c = c-num

    q = 'I am relaxed most of the time.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    n = n+num
```

```
530
531         q = 'I have difficulty understanding abstract ideas.'
532         while (True):
533             try:
534                 num = int(raw_input(q + '\n'))
535             except ValueError:
536                 print 'Wrong input, please try again'
537                 continue
538             if (num<1 or num>5):
539                 print 'Value must be in the range 1-5: Try again.'
540                 continue
541             break
542         o = o-num
543 
544         q = 'I feel comfortable around people.'
545         while (True):
546             try:
547                 num = int(raw_input(q + '\n'))
548             except ValueError:
549                 print 'Wrong input, please try again'
550                 continue
551             if (num<1 or num>5):
552                 print 'Value must be in the range 1-5: Try again.'
553                 continue
554             break
555         e = e+num
556 
557         q = 'I insult people.'
558         while (True):
559             try:
560                 num = int(raw_input(q + '\n'))
561             except ValueError:
562                 print 'Wrong input, please try again'
563                 continue
564             if (num<1 or num>5):
565                 print 'Value must be in the range 1-5: Try again.'
566                 continue
567             break
568         a = a-num
569 
570         q = 'I pay attention to details.'
571         while (True):
572             try:
573                 num = int(raw_input(q + '\n'))
574             except ValueError:
575                 print 'Wrong input, please try again'
576                 continue
577             if (num<1 or num>5):
578                 print 'Value must be in the range 1-5: Try again.'
579                 continue
580             break
581         c = c+num
582 
583 
584         q = 'I worry about things.'
585         while (True):
586             try:
587                 num = int(raw_input(q + '\n'))
588             except ValueError:
589                 print 'Wrong input, please try again'
590                 continue
```

```
591             if (num<1 or num>5):
592                 print 'Value must be in the range 1-5: Try again.'
593                 continue
594             break
595         n = n-num
596 
597         q = 'I have a vivid imagination.'
598         while (True):
599             try:
600                 num = int(raw_input(q + '\n'))
601             except ValueError:
602                 print 'Wrong input, please try again'
603                 continue
604             if (num<1 or num>5):
605                 print 'Value must be in the range 1-5: Try again.'
606                 continue
607             break
608         o = o+num
609 
610         q = 'I keep in the background.'
611         while (True):
612             try:
613                 num = int(raw_input(q + '\n'))
614             except ValueError:
615                 print 'Wrong input, please try again'
616                 continue
617             if (num<1 or num>5):
618                 print 'Value must be in the range 1-5: Try again.'
619                 continue
620             break
621         e = e-num
622 
623 
624         q = 'I sympathize with others feelings.'
625         while (True):
626             try:
627                 num = int(raw_input(q + '\n'))
628             except ValueError:
629                 print 'Wrong input, please try again'
630                 continue
631             if (num<1 or num>5):
632                 print 'Value must be in the range 1-5: Try again.'
633                 continue
634             break
635         a = a+num
636 
637 
638         q = 'I make a mess of things.'
639         while (True):
640             try:
641                 num = int(raw_input(q + '\n'))
642             except ValueError:
643                 print 'Wrong input, please try again'
644                 continue
645             if (num<1 or num>5):
646                 print 'Value must be in the range 1-5: Try again.'
647                 continue
648             break
649         c = c-num
650 
651         q = 'I seldom feel blue.'
```

```
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    n = n+num

    q = 'I am not interested in abstract ideas.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    o = o-num


    q = 'I start conversations.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    e = e+num

    q = 'I am not interested in other peoples problems.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    a = a-num


    q = 'I get chores done right away.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
```

```
713                 print 'Value must be in the range 1-5: Try again.'
714                 continue
715             break
716         c = c+num
717 
718         q = 'I am easily disturbed.'
719         while (True):
720             try:
721                 num = int(raw_input(q + '\n'))
722             except ValueError:
723                 print 'Wrong input, please try again'
724                 continue
725             if (num<1 or num>5):
726                 print 'Value must be in the range 1-5: Try again.'
727                 continue
728             break
729         n = n-num
730 
731 
732         q = 'I have excellent ideas.'
733         while (True):
734             try:
735                 num = int(raw_input(q + '\n'))
736             except ValueError:
737                 print 'Wrong input, please try again'
738                 continue
739             if (num<1 or num>5):
740                 print 'Value must be in the range 1-5: Try again.'
741                 continue
742             break
743         o = o+num
744 
745         q = 'I have little to say.'
746         while (True):
747             try:
748                 num = int(raw_input(q + '\n'))
749             except ValueError:
750                 print 'Wrong input, please try again'
751                 continue
752             if (num<1 or num>5):
753                 print 'Value must be in the range 1-5: Try again.'
754                 continue
755             break
756         e = e-num
757 
758         q = 'I have a soft heart.'
759         while (True):
760             try:
761                 num = int(raw_input(q + '\n'))
762             except ValueError:
763                 print 'Wrong input, please try again'
764                 continue
765             if (num<1 or num>5):
766                 print 'Value must be in the range 1-5: Try again.'
767                 continue
768             break
769         a = a+num
770 
771 
772           q = 'I often forget to put things back in their proper
place.'
```

```
773         while (True):
774             try:
775                 num = int(raw_input(q + '\n'))
776             except ValueError:
777                 print 'Wrong input, please try again'
778                 continue
779             if (num<1 or num>5):
780                 print 'Value must be in the range 1-5: Try again.'
781                 continue
782             break
783         c = c-num
784 
785         q = 'I get upset easily.'
786         while (True):
787             try:
788                 num = int(raw_input(q + '\n'))
789             except ValueError:
790                 print 'Wrong input, please try again'
791                 continue
792             if (num<1 or num>5):
793                 print 'Value must be in the range 1-5: Try again.'
794                 continue
795             break
796         n = n-num
797 
798 
799         q = 'I do not have a good imagination.'
800         while (True):
801             try:
802                 num = int(raw_input(q + '\n'))
803             except ValueError:
804                 print 'Wrong input, please try again'
805                 continue
806             if (num<1 or num>5):
807                 print 'Value must be in the range 1-5: Try again.'
808                 continue
809             break
810         o = o-num
811 
812         q = 'I talk to a lot of different people at parties.'
813         while (True):
814             try:
815                 num = int(raw_input(q + '\n'))
816             except ValueError:
817                 print 'Wrong input, please try again'
818                 continue
819             if (num<1 or num>5):
820                 print 'Value must be in the range 1-5: Try again.'
821                 continue
822             break
823         e = e+num
824 
825         q = 'I am not really interested in others.'
826         while (True):
827             try:
828                 num = int(raw_input(q + '\n'))
829             except ValueError:
830                 print 'Wrong input, please try again'
831                 continue
832             if (num<1 or num>5):
833                 print 'Value must be in the range 1-5: Try again.'
```

```
            continue
        break
    a = a-num


    q = 'I like order.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    c = c+num


    q = 'I change my mood a lot.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    n = n-num


    q = 'I am quick to understand things.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    o = o+num

    q = 'I do not like to draw attention to myself.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    e = e-num

    q = 'I take time out for others.'
    while (True):
```

```
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    a = a+num


    q = 'I shirk my duties.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    c = c-num


    q = 'I have frequent mood swings.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    n = n-num


    q = 'I use difficult words.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    o = o+num

    q = 'I do not mind being the center of attention.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
```

```
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    e = e+num


    q = 'I feel others emotions.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    a = a+num


    q = 'I follow a schedule.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    c = c+num


    q = 'I get irritated easily.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    n = n-num


    q = 'I spend time reflecting on things.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    o = o+num

```

```
    q = 'I am quiet around strangers.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    e = e-num

    q = 'I make people feel at ease.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    a = a+num


    q = 'I am exacting in my work.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    c = c+num

    q = 'I often feel blue.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
        if (num<1 or num>5):
            print 'Value must be in the range 1-5: Try again.'
            continue
        break
    n = n-num


    q = 'I am full of ideas.'
    while (True):
        try:
            num = int(raw_input(q + '\n'))
        except ValueError:
            print 'Wrong input, please try again'
            continue
```

```
1078            if (num<1 or num>5):
1079                print 'Value must be in the range 1-5: Try again.'
1080                continue
1081            break
1082        o = o+num
1083
1084
1085
1086        agreeableness_final = (24 + a)/10.0
1087        openness_final = (18 + o)/10.0
1088        conscientiousness_final =(24 + c)/10.0
1089        extraversion_final = (30 + e)/10.0
1090        neuroticism_final = (48 + n)/10.0
1091        print '\n\n'
1092
1093        print "Extraversion=", extraversion_final
1094
1095        print "Agreeableness=", agreeableness_final
1096
1097        print "conscientiousness=", conscientiousness_final
1098
1099        print "neuroticism=", neuroticism_final
1100
1101        print "openness=", openness_final
1102        print '\n'
1103
1104
1105        print ("THANK YOU FOR FILLING THE BIG FIVE QUESTIONAIRE. ")
1106        break
1107 #execfile('Analyzer.py')
1108 print 'Please Wait . . .'
1109
1110
1111
1112
1113
1114
1115
1116
myFiveFactorDatabase=pd.read_csv(FiveFactorFile,sep=",",header=None,
1117                                                    names=['MOVIE
GENRE','OPE','CON','EXT','AGR','NEU'], usecols=[0,1,2,3,4,5])
1118
1119
1120 myFiveFactorDatabase.loc[:, 'OPE']-=openness_final
1121  myFiveFactorDatabase.loc[:,  'OPE'] = myFiveFactorDatabase.loc[:,
'OPE'].abs()
1122 myFiveFactorDatabase.loc[:, 'CON']-=conscientiousness_final
1123  myFiveFactorDatabase.loc[:,  'CON'] = myFiveFactorDatabase.loc[:,
'CON'].abs()
1124 myFiveFactorDatabase.loc[:, 'EXT']-=extraversion_final
1125  myFiveFactorDatabase.loc[:,  'EXT'] = myFiveFactorDatabase.loc[:,
'EXT'].abs()
1126 myFiveFactorDatabase.loc[:, 'AGR']-=agreeableness_final
1127  myFiveFactorDatabase.loc[:,  'AGR'] = myFiveFactorDatabase.loc[:,
'AGR'].abs()
1128 myFiveFactorDatabase.loc[:, 'NEU']-=neuroticism_final
1129  myFiveFactorDatabase.loc[:,  'NEU'] = myFiveFactorDatabase.loc[:,
'NEU'].abs()
1130
1131 numbah = myFiveFactorDatabase.sum(axis=1, numeric_only=True)
```

```
numbahNames = myFiveFactorDatabase.loc[:, 'MOVIE GENRE']

genreSum=pd.concat([numbahNames, numbah], axis=1)

normalizedTable=RatingsNormalized(userID)




tab = knnPersonality5050(genreSum, myItemDatabase, 1682, 11, normalizedTable)



tab2 = knnPersonality8020(genreSum, myItemDatabase, 1682, 11, normalizedTable)



print nearestNeighbourPredictions(userID, neighbours)


#print 'Your highest rated movies are: \n', myBestMovies(userID, 10)
l5050 = finalNRecommendationsPersonality(tab, userID, 10)
print '50/50 recomendations:'
for i in l5050:
    print(i)
print '\n\n'
lknn = finalNRecommendations(userID, 10, 1682, 11) #knn
print 'KNN recomendations:'
for i in lknn:
    print(i)
print '\n\n'
l8020 = finalNRecommendationsPersonality(tab2, userID, 10) #80/20
print '80/20 recomendations:'
for i in l8020:
    print(i)
print '\n\nPress Enter to Exit . . .'
useless = (raw_input())

```