Isaak Christidis

Group, Fair and Graph-based Recommendations

Thesis in Computer Science
Supervisor: Panagiotis Symeonidis
August 2024

University of Aegean
Information and Communication Systems Department

Abstract

Recommendation systems or recommender systems is a undisputed huge part of the world wide web, being embedded in thousands of websites. The purpose of these systems is the collection and processing of different kind of statistics so that suggestions can be generated with the highest probability of the user to like them. Websites like Netflix.com, Amazon.com and Youtube.com depend on these systems and invest a big portion of their income to develop and evolve them so that ideal suggestions on movies, products and videos can be presented to users.
Recommendation systems main characteristic is the collection of data from users. These kind of systems collect elements like movies user have watched, products he have bought, his location and/or locations he has been to, products he had viewed or even reviewed and generally all kinds of interactions the user had with the product of each provider so that a collection of interaction can be generated. Afterwards, the system having all this information collected can recommend similar products with a higher purpose of the user re-buying. This paper will present and explain recommendation systems and update the reader on recent advancements that have been made on recommendation systems. There will be an analysis of recommendation systems with the help of some examples and some techniques used so that they can achieve better results. After that, this paper is divided into 4 sections which represent 4 categories of recommendation systems that will each be given a detailed analysis.

Preface

I would like to thank my professor for helping me complete this thesis with success and my family for always believing in me.

Contents

List of Figures

Chapter 1

Introduction

1.1 What is a recommendation system

What pushes us to buy a product? Usually a recommendation from a friend, an advert, a recommendation from the store employee or even the need for a product that pushed us searching for it. This procedure is being automated by recommendation systems, offering the user recommendations for items.
Recommendation parameters vary, depending on different factors. User likes are a main factor like when a user has declared in his profile information that he likes outside activities or even if he has rated an item. This rating can be general, like the first situation or a specific like a rating for a product in a range from one to five. Another general rating is a simple product purchase or trying a product. Trying out a product shows an interest in this product just like buying it in a bigger manner. These kind of ratings are constantly collected by the word wide web, having a big advantage of not requiring user's extra time.
In general, recommendation systems use resources from every user interaction with the system to recommend a product that according to user interactions is more possible to buy. An important exception is the systems that are based on knowledge, which take into consideration user likes that the user have set and not his history.
For example, if a user buys a compass then it is much more likely in a future purchase to buy a tent or boots. For knowledge based systems, a future purchase of a user that has declared to be a personal trainer is more like to be of an exercise tool or for 2 users that have declared their age to be 15 and 50 a future purchase to be headphones and glasses accordingly and not the other way around. Finally, it is obvious that the more interactions the user has with the system, the better the dataset and eventually the better

the recommendation will be.

1.2 Recommendation system goals

Recommendation systems goal is the increased number of sales. Main user of these systems are companies, which main goal is to increase profit. This makes recommendation systems the undisputed winner, being able to recommend carefully selected products/services and bringing the end user items that would help the system owner make a sale. With this privilege they achieve an increase in sales while using a very small amount of resources, increasing system's owner income and profit.
To help achieve this end goal, recommendation systems are setting small targets which areAggarwal, 2016:

1. Relativity
   Basic target of a recommendation system is the suggestion of products that are as close to each individual like as possible. Most of the times though, this is not adequate as can be seen below.

2. Innovation
   For a recommendation system to be successful, it is very important to suggest items that the user have not interact with. For example, in an online music streaming service, if a user has declared that he like a specific music type, the user shall not recommend his the most popular song of that genre. An action like that would have a negative effect when trying to push a user in a future purchase, making the system less profitable.

3. Coincidences
   A very important aspect of recommendation systems is the element of . User is greatly satisfied when he is overcome with the feeling of discovery of a product or service that hasn't come in contact before. This goal is different than relativity since it includes products that the user hasn't interact with, which truly surprise the user, in contrast with the items that are similar and haven't been discovered yet. For example, if a user have previously bought working out equipment, suggesting him training shoes contains the relativity element, but a recommendation of diving lenses contain the element of relativity while also trying to suggest a new category of interest, which is the main advantage of this goal. Suggesting the user something not so relevant does include the

risk of a useless recommendation, but there is a chance to broaden user interests which might result in future purchases of that product type

4. Variety
   Recommendation systems algorithms tend to suggest a list of the top products that satisfy the user interests. This has the disadvantage of suggesting the same items over and over to the same user, boring the user. In contrast, suggesting products that are not on the top product list gives the user a variety of choices, which leads to a new sell. In general, the system should also recommend items that are not in the user top recommendation list so that the user does not get bored getting shown the same thing over and over.

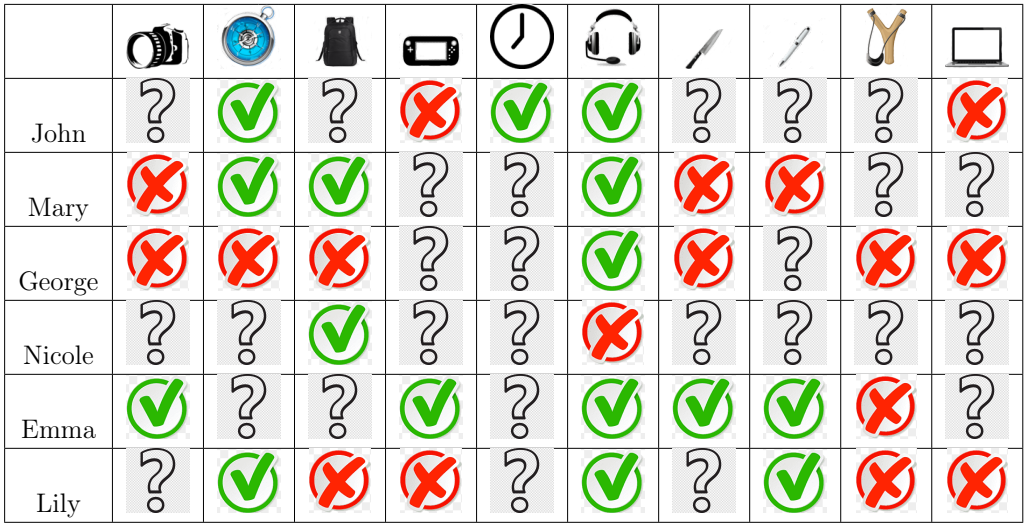
Smaller goals obviously do exist for recommendation systems, like the general satisfaction of the user. When a user gets recommendation more related to him he gets a more smooth and satisfying experience with the website, making it more possible to keep using it. Furthermore, the recommendation systems give the company insights for user interests and his next moves, giving her the opportunity to adjust the algorithm so that it gives the user a better experience. For instance, when it is observed that a suggestion outcomes to a purchase of this item for only one group of user, the system can be adjusted to prevent or mitigate the suggestion of this item to different groups. Likewise if it notices that after a purchase of an item, many users proceed to the purchase of a specific item, an offer can be made for a bundle of these two items.

1.3 Example

In this section an example will be presented to make many aspects of this paper easily understandable. In the following array some ratings of different users can be seen for a selection of products. It should be noted that the user data are not based on real data, but instead trying to represent a part of real ones.

1.4 Recommendation system categories

Basic recommendation systems models use two types of data, user interactions with items which is called collaborative filtering and user characteris-



tics or items which is called content based recommendations. Collaborative filtering contains user actions like ratings or purchases while content recommendations contain user profile or similar keywords. It should be mentioned that because content based recommendation systems are exclusively using user info and characteristics, making them weak and boring, they are usually used in combination with collaborative filtering. The effort to mitigate the negative features of these two models as well as the evolution of technology has led to the creation of additional hybrid models in order to exploit the positive features and reduce the negative ones such as re-selected recommendations or boring and expected recommendations. Additional techniques such as knowledge-based systems and demographic recommendation systems have been created and developed.

1.4.1 Collaborative filteriing

Collaborative filtering is the use of multiple users' interactions to produce a prediction for users who have not interacted with this product. For example, if Mary and John have similar preferences then it is very likely that a product that John has not interacted with (bought) and Mary has rated as good, John will like. This model uses the ratings that users have given to products/services as a key element. The most common way of rating is by giving a ranking rating from 1 to 5 with 1 being the worst and 5 being the best. Other forms of ratings are positive/negative reviews, simply promoting a product or buying it
The main challenge of collaborative filtering systems is that a large propor-

tion of the data used to extract recommendations is empty. For example, in a database containing movie or product reviews, users will have seen or purchased a very small percentage of the total number of products offered. For John and Mary's example, the product that John would not have reviewed would be blank in the database. Filling these gaps is the purpose of algorithms that use collaborative filtering. In John and Mary's example, when the algorithm finds some commonalities between the two, it will understand that there is a relationship between them and will try to fill in the gaps using the other person as a model.
There are mainly two methods that utilize collaborate filtering, the ones based on memory and the ones based on model.

1.4.1.1 Memory based collaborate filtering

Memory-based methods are sometimes referred to as cooperative neighbor-based filtering algorithms. These methods are among the first collaborative filtering methods in which the recommendation is based on neighbors, which can either be users who have similar preferences or products that are similar to one the user likes.

1. User-based collaborate filtering
   In this category, reviews given by users with similar preferences are used to make the recommendation to the user. Thus, a key objective is to find users who have similar preferences to the user. In the example with John and Mary, the computer seeing that the reviews of the compass and the headphones are both positive from both will consider these two users to have similar preferences and will recommend John the bag that Mary liked.
   One of the main problems of collaborative user-based algorithms is user bias. Some users are biased to evaluate items they like only while others items they don't like only. Because of this, users' ratings usually follow a certain pattern, which makes the algorithm's job difficult because combined with the fact that most of the table of ratings between users and objects is empty. For example, the algorithm will perceive that users Mary, John and Lily have common preferences since all three like the compass and headphones. However, if they notice the intersection of the ratings of these three users contains only these two items since there is even one missing rating from one of them on the other products.
   One method that detects the similarity of ratings between users is the

Pearson correlation coefficient method. This method works on the intersection of two users' ratings. If John's ratings are all the ratings he has made, the completed items in his row would be:

$$I_J = \{2, 4, 5, 6, 10\} \tag{1.1}$$

and Lily's ratings would be:

$$I_L = \{2, 3, 4, 6, 8, 9, 10\} \tag{1.2}$$

then the users John and Lily set of ratings will be the cross-section of theses two sets, a set of 4.

$$I_{J,L} = I_J \cap I_L = \{2, 4, 6, 10\} \tag{1.3}$$

The first step of the method is to find the average of the intersection of the reviews of the two users' ratings. The Pearson method requires finding the average at the intersection of the reviews, which is extremely difficult in a recommendation system because, as mentioned above, most of the ratings table in a real system is usually empty. Thus, the average is calculated separately for each user, which makes this method less reliable but more information-rich. For example, a user's average would be:

$$M_x = \frac{\sum_{i \in I_x} r_{xi}}{|I_x|} \tag{1.4}$$

Which represents the sum of the users ratings devided by the number of ratings, which for John would be:

$$M_g = \frac{\sum\{2, 1, 2, 2, 1\}}{5} = 8/5 = 1.6. \tag{1.5}$$

We are setting a rating of 2 for a positive rating while a 1 for a negative one. After this, to find the similarity between two users (x,y) we will use the Pearson equation:

$$Sim_{x,y} = \frac{\sum_{i \in I_x \cap I_y} (r_{xi} - \mu_x)(r_{yi} - \mu_y)}{\sqrt{\sum_{i \in I_x \cap I_y} (r_{xi} - \mu_x)^2} \sqrt{\sum_{i \in I_x \cap I_y} (r_{yi} - \mu_y)^2}}. \tag{1.6}$$

For example, to find the similarity between John and Lily, we have to find the mean value of Lilys ratings:

$$M_k = \frac{\sum\{2, 1, 1, 2, 2, 1, 1\}}{7} = 10/7 = 1.42. \tag{1.7}$$

So if we apply Pierson's formula of similarity to Jonh and Lily we get the result:

$$Sim_{x,y} = \frac{\sum_{i \in I_x \cap I_y}(r_{xi} - \mu_x)(r_{yi} - \mu_y)}{\sqrt{\sum_{i \in I_x \cap I_y}(r_{xi} - \mu_x)^2}\sqrt{\sum_{i \in I_x \cap I_y}(r_{yi} - \mu_y)^2}} =$$
$$\frac{(0.4)(0.58) + (-0.6)(-0.42) + (0.4)(0.58) + (-0.6)(-0.42}{(\sqrt{0.4)^2 + (-0.6)^2 + (0.4)^2 + (-0.6)^2}\sqrt{0.58^2 + (-0.42)^2 + (0.58)^2 + (-0.42)^2}} =$$
$$\frac{0.23 + 0.25 + 0.23 + 0.25}{\sqrt{0.16 + 0.36 + 0.16 + 0.36}\sqrt{0.33 + 0.17 + 0.33 + 0.17}}.$$
$$= \frac{0.96}{\sqrt{1.04}\sqrt{1}} = \frac{0.96}{1.02 \cdot 1} = \frac{0.96}{1.08} = 0.94 \tag{1.8}$$

While if we apply the same equation to John and Mary, we are getting:

$$M_m = \frac{\sum\{1, 2, 2, 2, 1, 1\}}{6} = 9/6 = 1.5. \tag{1.9}$$

$$Sim_{x,y} = \frac{\sum_{i \in I_x \cap I_y}(r_{xi} - \mu_x)(r_{yi} - \mu_y)}{\sqrt{\sum_{i \in I_x \cap I_y}(r_{xi} - \mu_x)^2}\sqrt{\sum_{i \in I_x \cap I_y}(r_{yi} - \mu_y)^2}} =$$
$$\frac{(2 - 1.6)(2 - 1.5) + (2 - 1.5)(2 - 1.6)}{\sqrt{(2 - 1.6)^2 + (2 - 1.6)^2}\sqrt{(2 - 1.5)^2 + (2 - 1.5)^2}} = \tag{1.10}$$
$$= \frac{0.2 + 0.2}{\sqrt{0.16 + 0.16}\sqrt{0.25 + 0.25}} =$$
$$= \frac{0.4}{\sqrt{0.32}\sqrt{0.5}} = \frac{0.4}{0.56 \cdot 0.7} = \frac{0.4}{0.39} = 1$$

So there is a greater similarity between John's and Mary's reviews than the reviews between John and Lily's with a slight discrepancy of course. This is because although the two reviews in the intersection of John and Mary's reviews overlap there is a larger sample for John and Lily's reviews.

2. Item-based Collaborative filtering

   In this category objects are used to make the recommendation. First, a group of objects that are similar to each other should be found. Then, the reviews for the other objects will be used to decide whether the

user will like the object. In our example, the algorithm will see that user John has positively rated the compass and the clock and will suggest the sling, placing them all in the "outdoor" category. Pearson's method mentioned above can be used to find the similarity between the objects(columns), but the adjusted covariance method yields better results. In this method, the values of the criteria must first be adjusted using the method mean centering. Top calculate this, the average of the user's reviews is found first and then subtracted from each review. Thus the table of reviews will become: And so by applying the mean-

											Mean Value
John		2		1	2	2				1	1.6
Mary	1	2	2			2	1	1			1.5
Peter	1	1	1			2	1		1	1	1.14
Nicole			2			1					1.5
Emma	2			2		2	2	2	1		1.83
Lily		2	1	1		2		2	1	1	1.42

centering method, by subtracting the mean from each rating value, the rating table will become: Essentially an algorithm is created:

										
John		0.4		-0.6	0.4	0.4				-0.6
Mary	-0.5	0.5	0.5			0.5	-0.5	-0.5		
Peter	0	0	0				0		0	0
Nicole										
Emma	0			0		0	0	0	0	
Lily		0.6	-0.4	-0.4		0.6		0.6	-0.6	-0.6

This can be easily done with the code given in section 9.1.3. , specifically in the sub-chapter "Calculating the central value of ratings" the code for calculating the average of each user is given. Then in the subsection "Calculating the centroid value" is given to subtract each user's average from each review.

1.4.2 Content-based filtering

As it was previously mentioned, collaborative filtering uses other users' ratings to make a recommendation without taking into account the items' char-

acteristics . This is extremely flawed for the system since for example Mary likes both the bag and the compass which means there is a good chance she likes either the watch or the camera since you fall into the "outdoor hobby" category. In such cases as in cases where an object is new or has few reviews content-based filtering algorithms are more effective and reliable.
The algorithms used for content-based recommendations rely on two types of data for their recommendations, reviews and user profile. For example, Chris has positively rated a watch and a computer, two electronic items for the home. The algorithm could recommend for Chris the electronic video game. As shown in the figure, content-based algorithms try to find a link between items that the user has already interacted with other products to recommend them to the user.

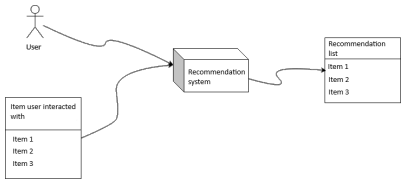


Figure 1.4.1: Content Based Filtering(draw.io)

The characteristics of content-based algorithms provide them with some advantages over collaborate filtering algorithms. Their main one is their high tolerance to cold start. When a system or product is new there are not enough user reviews and as a result the collaborative filtering based algorithm provides incomplete recommendations. In contrast, content-based algorithms do not rely on other users' reviews at all and yield better results in these cases. Still, since these algorithms are based on product descriptions, richer descriptions also make these algorithms more effective. Thus, these algorithms proved to be more effective in systems where products contain a lot of content, such as in recommendations of articles or even web pages.
However, content-based systems have some disadvantages such as:

1. Weakness in media products
   Systems find it difficult to identify features in media so as to recommend similar products. Recommendations in a media system can be made either by using the description and characteristics of the media or by suggesting randomly.

2. Great emphasis on the user profile

3. These systems rely heavily on the characteristics set by the user when creating them. Thus, if the user interacts with a product that is not

directly related to features of his profile, the results of the recommendation system will be incomplete. To solve this problem many use random suggestions to populate the recommendation list

We can see that content-based algorithms are based on object descriptions as well as user attributes. For the reason that the recommendation generation is done by a computer the descriptions of the items and the user's attributes have to be converted into values that a computer understands. The same has to be done with text-based products. From this text keywords have to be extracted to make categorisation and subsequent recommendation easier. This process can be broadly divided into the following three steps:

1.4.2.1 Pre-processing and attribute extraction

Content-based systems are used in various types of products such as web pages, news articles, music, product contents and this information should be converted into a representation system as vectors in space which represent keywords. It is important to emphasize the importance of selecting the most information rich features. This step is highly depending on the type of system for which recommendations will be made.
First, the extraction of the object attributes is performed. The most common way is to extract keywords from the products of the system as it is the most common way of describing a product. For example, in a news website each article has a title, publisher, description as well as text from which keywords can be extracted, most of which are highly correlated and represent the content of the text/article well. In some cases that the content and characteristics of products can be more satisfactorily represented by numerical units such as price, colour, size, a multidimensional representation of these is more satisfactory and effective. After these elements are extracted either as keywords or as some multidimensional representation, a weight should be given to each value which will indicate how important that element is and how much weight the recommendation algorithm will give to it. For example, in a book or a movie, more weight will be given to the title, author or gender of the movie respectively than to a word within the text or a dialogue within the movie. Then these extracted pluralities will have to be represented in a way that is processable. This means that the keywords should be structured so that they can then be processed in order to create recommendations. This step is done in three phases, removing redundant words, cutting off endings and extracting phrases.

1. Remove unnecessary words

In the stage of removing unnecessary words, words that do not help the recommendation system are removed. Words such as articles, prepositions, pronouns and links are considered words that do not help in the content of the text and in the creation of recommendations are removed.

2. Suffix cutoff

   In this stage, grouping of words with the same content takes place. A word has the same content in either singular or plural. Such words as well as verb genders are merged at the root, which can be negative if not managed properly. For example, if in an article the word photovoltaic has a different meaning for a house if we group it into the words light and volt.

3. Phrase extraction
   In this stage, words that usually appear together are grouped together. For example. the phrase electric panel has a different meaning than the words individually.

These words are then represented in a space as vectors along with their frequency of occurrence. Using their frequency of occurrence is not effective, however, since statistically the words that appear many times are usually not discriminate. You choose to represent their frequency f_i as the logarithm of the number found in text n to the number of texts where this word has been found n_i

$$f_i = \log(\frac{n}{n_i}) \tag{1.11}$$

It should be mentioned at this point the cases of "malicious" unwanted multiple words (spam) in which, once detected, a reduction of their number can be done simply by using any reduction method such as root, logarithm or division. In the next step, the user's preferences are collected by gathering data such as objects with which the user has interacted. Such statistics can be either objects that the user has evaluated, purchases and observations that the user has made, some written evaluation or experiences that the user has created. In the last step of feature extraction, feature words are selected and assigned weights. The general purpose of this step is to avoid or isolate words that do not have important information, similar to the first step of keyword extraction and frequency assignment. The basic difference between this and the first step is the user factor. While in the first step no importance is given to the user ratings, in this step significant importance is given to the

user ratings. There are five methods chosen for this step, of entropy, Gini index, statistical square, deviation and feature weight.

1.4.2.2 User learning

In this step a user model is created to calculate the user's interests in products with the help of the user's previous interactions with the system. These interactions are used in conjunction with the characteristics of various products to create the learning data and then using this learning data you create the learning model. This model is also the user model since it creates a relationship between the user's interests and product attributes.
This step is similar to the data class and regression problem depending on whether the user data is of the binary or scale form respectively. The user's reviews for each text are used to build the user's model which represents the user's profile. Techniques used in knowledge-based filtering and collaborative filtering are used with some variations in this step. Methods like nearest neighbor class, case based in knowledge based filtering, Bayesian based class, rule based class, regression based model are proved to be very efficient for this step.

Chapter 2

Group recommendations

Group recommendation-based recommendation systems use algorithms to generate recommendations to a set of users (groups). Because these groups will often buy the service as a whole, a product must be found based on a set of individuals/users. Products of such recommendations are movies, series, TV programs, music songs or travel packages.
This coefficient seems to be just an average of the other recommendations, but in cases such as the spread of emotions and social conformity it has been shown to be unsatisfactory.
An important area of group recommendation systems is social networking sites, where you use the idea of a group to display posts, suggest networking with another user, or even buying a product to people who join a group. A key driver of these recommendations within social media is user location as it provides important information about user preferences. In particular, one solution to the phenomenon of merging recommendations into one group while each user has a different influence on the decision of the final product promoted is to submit an article /cite11 that uses the recommendations and preferences of each user individually. This method combined with the use of average weight yielded positive results making the system more successful. This phenomenon of spreading positive and negative emotions to the group, and social conformity like individual conformity to the preferences of the group, are differently confronted in each of the three models mentioned above. In the models of collaborative filtering and content-based filtering are similar to the techniques used in the simple systems while there are some differences for the knowledge-based models.

2.1 Group recommendations based on collaborative filtering and content-based filtering

Group recommendation systems based on collaborative filtering and content-based filtering is the same and is based on two stepsZafarani, 2014:

1. Create a recommendation separately for each user and calculate a rating for each user and product combination in a subset of products for a group of users.

2. For each item, aggregate the ratings of the group members that were created into one value using either an average weight function, a general like function, or a combination of the two.

These functions each have their own way of implementation:

2.1.1 Kemeny-optimal function

In general like functions the recommendation score is the lowest rating of each group user, i.e., the rejection of any proposal that has even one negative rating by a group user. As can be easily noticed, this method is ideal for avoiding the phenomenon of spreading sentiment.

1. Pierson function
   For example, we will compute predictions for the group of John, Mary and Kate based on collaborative user-based filtering with a general liking function. Initially the table of these three is as shown below:

										
John		2		1	2	2				1
Mary		1			2			1		2
Kate		2	1	1		2		2	1	1

Then we need to calculate the similarity between users to be able to find for each of them the three users that has the highest correlation. So, first we need to calculate the average of all the users.

$$M_1 = \frac{2+1+2+2+1}{5} = 1.6 \tag{2.1}$$

$$M_2 = \frac{1+2+1+2}{4} = 1.5 \tag{2.2}$$

$$M_3 = \frac{1+1+1+2+1+1+1}{7} = 1.14 \tag{2.3}$$

$$M_4 = \frac{2+1}{2} = 1.5 \tag{2.4}$$

$$M_5 = \frac{2+2+2+2+2+1}{6} = 1.83 \tag{2.5}$$

$$M_6 = \frac{1+2+1+2}{4} = 1.5 \tag{2.6}$$

$$M_7 = \frac{1+1+1+2+2}{5} = 1.4 \tag{2.7}$$

$$M_8 = \frac{2+1+1+2+2+1+1}{7} = 1.42 \tag{2.8}$$

And then we calculate the similarities

$$\begin{aligned} Sim_{1,2} = \frac{\sum_{i \in I_x \cap I_y}(r_{xi} - \mu_x)(r_{yi} - \mu_y)}{\sqrt{\sum_{i \in I_x \cap I_y}(r_{xi} - \mu_x)^2}\sqrt{\sum_{i \in I_x \cap I_y}(r_{yi} - \mu_y)^2}} = \\ \frac{(0.4 * (0.5)) + (0.4 * 0.5)}{\sqrt{0.4^2 + 0.4^2}\sqrt{0.5^2 + 0.5^2}} = \\ = \frac{0.2 + 0.2}{\sqrt{0.16 + 0.16}\sqrt{0.25 + 0.25}} = \\ = \frac{0.4}{\sqrt{0.32}\sqrt{0.5}} = \frac{0.4}{0.56 * 0.7} = -\frac{0.4}{0.39} = 1 \end{aligned} \tag{2.9}$$

In the same way we calculate the correlations between all users.
For John:

$$Sim_{1,3} = 0.5 \tag{2.10}$$

$$Sim_{1,4} = -1 \tag{2.11}$$

$$Sim_{1,5} = -0.2 \tag{2.12}$$

$$Sim_{1,6} = 0.94 \tag{2.13}$$

After that we calculate Mary's:

$$Sim_{2,3} = \frac{\sum_{i \in I_x \cap I_y}(r_{xi} - \mu_x)(r_{yi} - \mu_y)}{\sqrt{\sum_{i \in I_x \cap I_y}(r_{xi} - \mu_x)^2}\sqrt{\sum_{i \in I_x \cap I_y}(r_{yi} - \mu_y)^2}} = \\ = 0.42 \tag{2.14}$$

$$Sim_{2,4} = 0 \tag{2.15}$$

$$Sim_{2,5} = -0.5 \tag{2.16}$$

$$Sim_{2,6} = 0.07 \tag{2.17}$$

And for Jack:

$$Sim_{3,4} = -0.81 \tag{2.18}$$

$$Sim_{3,5} = 0.27 \tag{2.19}$$

$$Sim_{3,6} = 0.07 \tag{2.20}$$

For Nicole:

$$Sim_{4,5} = -1 \tag{2.21}$$

$$Sim_{4,6} = -0.99 \tag{2.22}$$

For Georgia:

$$Sim_{5,6} = 0.53 \tag{2.23}$$

For Kate we already calculated all the similarities.
The same process can be done very easily using a Python programming language script as shown in the code in section 7.1.1 and in particular in the subsections "Table initialization and averaging" and "Calculating users' similarities". The code function can be seen in step 1 of is essentially doing the step 1 of GroupRecommendations pseudocode.

Algorithm 1 Step 1: Calculating user similarities

for $user1 \in users$ **do**
for $user2 \in users$ **do**
for $item \in items$ **do** $nominator = nominator + ((rating(user1, item)$ -$mean\_value(user1)$) * ($rating(user2,item)$ - $mean\_value(user2)))$
$denominator1 = denominator1 + (rating(user1, item)$ - $mean\_value(user1))^2$ $denominator2 = denominator2 + (rating(user2, item)$ - $mean\_value(user2))^2$ $recommendation = nominator/\ square\_root(denominator1) * square\_root(denominator2)$
end for
end for
end for

Algorithm 2 Step 2: Finding the top n neighbors

for $similarity \in user\_similarities$ **do**
if $similarity > minimum$ ($user \in user\_similarities$) **then**
$top\_similarities(user)($ $minimum$ ($user \in user\_similarities))$ $= similarity$
end if
end for

Algorithm 3 Step 3: Predicting user ratings

```
for user_ratings ∈ ratings_array do
    for rating ∈ user_ratings do
        if rating=0 then
            for top_similarity ∈ top_similarities do
                sum_similarities += top_similarity
                sum_nominator += user_rating(top_similarity) - mean_value(top_similarity)
            end for
            predicted_rating = sum_nominator / sum_similarities
        end if
    end for
end for
```

And so we create the table:

	John	Mary	Jack	Nicole	Georgia	Kate
John	1	1	0.51	-1	-0.2	0.94
Mary		1	0.42	0	-0.5	0.07
Jack			1	-0.81	0.27	0.07
Nicole				1	-1	-0.99
Georgia					1	0.53
Kate						1

So, if we select the three highest associations of users with others we get:

(a) For the user John, you observe a higher correlation with the users Mary, Jack and Kate. So we can also find the predictions for the products he is missing.

- For the camera:

$$
\begin{aligned}
r_{1,camera} = \mu + \frac{Sim_{1,2} * (r_2 - \mu_2) + Sim_{1,3} * (r_3 - \mu_3) + Sim_{1,8} * (r_8 - \mu_8)}{Sim_{1,2} + Sim_{1,3} + Sim_{1,8}} = \\
\mu + \frac{Sim_{1,2} * (1 - 1.5) + Sim_{(}1,3) * (1 - 1.14)}{Sim_{1,2} + Sim_{1,3}} = \\
= \mu + \frac{(-0.5) * 1 + 0.51 * (1 - 1.14)}{1 + 1.14} = \\
= 1.6 + \frac{(-0.5) - 0.07}{1 + 1.14} = 1.6 + \frac{-0.57}{2.14} = 1.6 - 0.26 = 1.34
\end{aligned}
\tag{2.24}
$$

- For backpack:

$$r_{1,bag} = \mu + \frac{Sim_{1,2} * (r_2 - \mu_2) + Sim_{1,3} * (r_3 - \mu_3) + Sim_{1,8} * (r_8 - \mu_8)}{Sim_{1,2} + Sim_{1,3} + Sim_{1,8}} = \mu + \frac{Sim_{1,2} * (2 - 1.5) + Sim_(1,3) * (1 - 1.14) + Sim_(1,8) * (1 - 1.42)}{Sim_{1,2} + Sim_{1,3} + Sim_{1,8}} = = \mu + \frac{(0.5) * 1 + 0.51 * (1 - 1.14) + 0.94 * (1 - 1.42}{1 + 1.14 + 0.94} = = 1.6 + \frac{0.5 - 0.07 - 0.394}{1 + 0.51 + 0.94} = 1.6 + \frac{0.46}{2.45} = 1.6 + 0.18 = 1.78 \quad (2.25)$$

- For the knife:

$$r_{1,knife} = \mu + \frac{Sim_{1,2} * (r_2 - \mu_2) + Sim_{1,3} * (r_3 - \mu_3) + Sim_{1,8} * (r_8 - \mu_8)}{Sim_{1,2} + Sim_{1,3} + Sim_{1,8}} = \mu + \frac{Sim_{1,2} * (1 - 1,5) + Sim_{1,3} * (1 - 1.14)}{Sim_{1,2} + Sim_{1,3}} = = \mu + \frac{0.5 * (-0.5) + 0.51 * (1 - 1.14)}{0.5 + 0.51} = \frac{-0.25 - 0.07}{1.01} = 1.6 - 0.3 = 1.3 \quad (2.26)$$

- For the pen:

$$r_{1,pen} = \mu + \frac{Sim_{1,2} * (r_2 - \mu_2) + Sim_{1,3} * (r_3 - \mu_3) + Sim_{1,8} * (r_8 - \mu_8)}{Sim_{1,2} + Sim_{1,3} + Sim_{1,8}} = \mu + \frac{Sim_{1,2} * (1 - 1.5) + Sim_{1,8} * (2 - 1.42)}{Sim_{1,2} + Sim_{1,8}} = \frac{1 * (-0.5) + 0.94 * (2 - 1.42)}{0.5 + 0.94} = 1.6 + \frac{-0.5 + 0.94 * 0.56}{1.44} = 1.6 + 0.01 = 1.61 \quad (2.27)$$

- For the slingshot:

$$r_{1,slingshot} = \mu + \frac{Sim_{1,2} * (r_2 - \mu_2) + Sim_{1,3} * (r_3 - \mu_3) + Sim_{1,8} * (r_8 - \mu_8)}{Sim_{1,2} + Sim_{1,3} + Sim_{1,8}} = \mu + \frac{Sim_{1,3} * (1 - 1.14) + Sim_{1,8} * (1 - 1.42)}{Sim_{1,3} + Sim_{1,8}} = 1.6 + \frac{0.51 * (-0.14) + 0.94 * (1 - 1.42)}{0.51 + 0.94} = 1.6 + \frac{-0.07 - 0.39}{1.45} = 1.6 - 0.31 = 1.29 \quad (2.28)$$

(b) For Mary you observe a higher correlation with the users John, Jack and Kate, so the calculations of her empty ratings will be based on them . We have for the average: While the correlations are:

And for its predictions it will apply:

- For the video game:

$$r_{1,videogame} = \mu + \frac{Sim_{1,2} * (r_2 - \mu_2) + Sim_{2,3} * (r_3 - \mu_3) + Sim_{2,8} * (r_8 - \mu_8)}{Sim_{1,2} + Sim_{2,3} + Sim_{2,8}} = 0.91 \tag{2.29}$$

- For the clock:

$$r_{1,clock} = \mu + \frac{Sim_{1,2} * (r_2 - \mu_2) + Sim_{1,3} * (r_3 - \mu_3) + Sim_{1,8} * (r_8 - \mu_8)}{Sim_{1,2} + Sim_{1,3} + Sim_{1,8}} = 1.91 \tag{2.30}$$

- For the slingshot:

$$r_{1,slingshot} = \mu + \frac{Sim_{1,2} * (r_2 - \mu_2) + Sim_{1,3} * (r_3 - \mu_3) + Sim_{1,8} * (r_8 - \mu_8)}{Sim_{1,2} + Sim_{1,3} + Sim_{1,8}} = 1.22 \tag{2.31}$$

- For the laptop:

$$r_{1,laptop} = \mu + \frac{Sim_{1,2} * (r_2 - \mu_2) + Sim_{1,3} * (r_3 - \mu_3) + Sim_{1,8} * (r_8 - \mu_8)}{Sim_{1,2} + Sim_{1,3} + Sim_{1,8}} = 1.04 \tag{2.32}$$

(c) For Kate you observe a higher correlation with the users John, Jack and Georgia, so the calculations of her empty ratings will be based on them

.

And for her predictions:

- For the camera:

$$r_{1,videogame} = \mu + \frac{Sim_{1,8} * (r_1 - \mu_1) + Sim_{3,8} * (r_3 - \mu_3) + Sim_{5,8} * (r_5 - \mu_5)}{Sim_{1,8} + Sim_{3,8} + Sim_{5,8}} = 1.26 \tag{2.33}$$

- For the clock:

$$r_{1,clock} = \mu + \frac{Sim_{1,8} * (r_1 - \mu_1) + Sim_{3,8} * (r_3 - \mu_3) + Sim_{5,8} * (r_5 - \mu_5)}{Sim_{1,8} + Sim_{3,8} + Sim_{5,8}} = 1.83 \tag{2.34}$$

For the pen:

-

$$r_{1,slingshot} = \mu + \frac{Sim_{1,8} * (r_1 - \mu_1) + Sim_{3,8} * (r_3 - \mu_3) + Sim_{5,8} * (r_5 - \mu_5)}{Sim_{1,8} + Sim_{3,8} + Sim_{5,8}} = 1.26 \tag{2.35}$$

Then the users with the biggest similarities need to be calculated. This can be done using the logic of step 2 of the pseudocode named "Finding the top n neighbors", which code is on chapter 9, section 2 Finally we can calculate the predictions like the step 3, "Predicting user rating" pseudocode shows up.

Then to merge the evaluations we will use the Kemeny-optimal method. In this method we also need a way of calculating distance Zafarani, 2014. The most widely used distance calculation method is the Kendal tau method in which we have the following evaluations:

	John	Mary	Kate
	B	C	A
	C	A	C
	A	B	B

Then they should be sorted in ascending order with respect to someone, e.g. John by converting the table :

	John	Mary	Kate
	A	B	B
	B	C	A
	C	A	C

And then we need to calculate for each possible combination of ratings its distance to the other combinations. For our example the possible pairs will be :

- ABC
- ACB
- BAC
- BCA
- CAB
- CBA

	ABC	ACB	BAC	BCA	CAB	CBA
ABC						
ACB						
BAC						
BCA						
CAB						
CBA						

And then we need to calculate the distance of each one to all the others, that is, we need to fill the table:

The way to calculate the distance is to compare the positions of each alphabetical possible combination and then subtract all the inconsistent ones from the consistent ones by the total number of combinations.

So, for our example:

- ABC-ACB
  For this pair we need to compare the positions of all alphabetically ordered combinations of possible ratings. That is, for the pairs A-B, A-C, B-C we need to find and compare their positions in each of ABC and ACB.
  . More specifically, the position of A is 0 in ABC and 0 in ACB while the position of B is 1 in ABC and 2 in ACB and $0<1$ and $0<2$ so this pair is considered congruent. The table shows all the possible pairs and their comparison.

A-B	A-B	Comparison
A<B	A<B	consistant
A<B	A>B	inconsistant
A>B	A>B	consistant
A>B	A<B	inconsistant
A=B	A>B	
A<B	A=B	

By doing the same procedure for A-C and B-C we can complete

the table:

	ABC	ACB	Comparison	Tau
A,B	A<B	A<B	consistent	
A,C	A<C	A<C	consistent	
B,C	B<C	B>C	inconsistent	
Tau				0.33

- BAC-ABC

	BAC	ABC	Caparison	Tau
A,B	A>B	A<B	inconsistent	
A,C	A<C	A<C	consistent	
B,C	B<C	B<C	consistent	
Tau				0.33

- BAC-ACB

	BAC	ACB	Comparison	Tau
A,B	A>B	A<B	inconsistent	
A,C	A<C	A<C	consistent	
B,C	B<C	B>C	inconsistent	
Tau				-0.33

- BCA-ABC

	BCA	ABC	Comparison	Tau
A,B	A>B	A<B	inconsistent	
A,C	A>C	A<C	inconsistent	
B,C	B<C	B<C	consistent	
Tau				-0.33

- BCA-ACB

	BCA	ACB	Comparison	Tau
A,B	A>B	A<B	inconsistent	
A,C	A>C	A<C	inconsistent	
B,C	B<C	B>C	inconsistent	
Tau				-1

- BCA-BAC

	BCA	BAC	Comporison	Tau
A,B	A>B	A>B	consistent	
A,C	A>C	A<C	inconsistent	
B,C	B<C	B<C	inconsistent	
Tau				0.33

- CAB-ABC

	CAB	ABC	Comparison	Tau
A,B	A<B	A<B	consistent	
A,C	A>C	A<C	inconsistent	
B,C	B>C	BC	inconsistent	
Tau				-0.33

- CAB-ACB

	CAB	ACB	Comparison	Tau
A,B	A<B	A<B	consistent	
A,C	A>C	A<C	inconsistent	
B,C	B>C	B>C	consistent	
Tau				0.33

- CAB-BAC

	CAB	BAC	Comparison	Tau
A,B	A<B	A>B	inconsistent	
A,C	A>C	A<C	inconsistent	
B,C	B>C	B<C	inconsistent	
Tau				-1

- CAB-BCA

	CAB	BCA	Comparison	Tau
A,B	A<B	A>B	inconsistent	
A,C	A>C	A>C	consistent	
B,C	B>C	B<C	inconsistent	
Tau				-0.33

- CBA-ABC

	CBA	ABC	Comparison	Tau
A,B	A>B	A<B	inconsistent	
A,C	A>C	A<C	inconsistent	
B,C	B>C	B<C	inconsistent	
Tau				-1

- CBA-ACB

	CBA	ACB	Comparison	Tau
A,B	A>B	A<B	inconsistent	
A,C	A>C	A<C	inconsistent	
B,C	B>C	B>C	consistent	
Tau				-0.33

- CBA-BAC

	CBA	BAC	Comparison	Tau
A,B	A>B	A>B	consistent	
A,C	A>C	A<C	inconsistent	
B,C	B>C	B<C	inconsistent	
Tau				-0.33

- CBA-BCA

	CBA	BCA	Comparison	Tau
A,B	A>B	A>B	consistent	
A,C	A>C	A>C	consistent	
B,C	B>C	B<C	inconsistent	
Tau				0.33

- CBA-CAB

	CBA	CAB	Comparison	Tau
A,B	A>B	A<B	inconsistent	
A,C	A>C	A>C	consistent	
B,C	B>C	B>C	consistent	
Tau				0.33

So, we can fill up the array:

	ABC	ACB	BAC	BCA	CAB	CBA
ABC	1					
ACB	0.33	1				
BAC	0.33	-0.33	1			
BCA	-0.33	-1	0.33	1		
CAB	-0.33	0.33	-1	-0.33	1	
CBA	-1	-0.33	-0.33	0.33	0.33	1

We then select for the pairs we want to compare the 2 or 5 largest values and the pair they represent. Specifically, in our example for the user ratings of John, Mary, Kate we select:

- For the next 2 pairs: In this method, selection is based on abun-

ABC	BAC	BCA
ACB	ABC	BAC
BAC	BCA	CBA

dance, i.e. selection is based on the number of pairs that appeared in our table. Thus, the adjacent table can be created with the number of each relationship:

Pair	Amount
ABC	1
ACB	1
BAC	2
BCA	1
CBA	1

So the BAC will be selected.

- For the 5 older couples: In this method you make a selection based on their value. This is the same procedure, except that the selection is based on the artifact of the elements. This is how the table is created:

ABC	BAC	BCA
ACB(0.33)	ABC(0.33)	ABC(-0.33)
BAC(0.33)	ACB(-0.33)	ACB(-1)
BCA(-0.33)	BCA(0.33)	BAC(0.33)
CAB(-0.33)	CAB(-1)	CAB(-0.33)
CBA(-1)	CBA(-0.33)	CBA(0.33)

So the sum array will be like: So again BAC will be chosen because of its higher value.

Pair	Crowd
ABC	0
ACB	-1
BAC	0.66
BCA	0
CBA	-1
CAB	-1.66

This process can be done again using the python programming language and in particular with the code of chapter 7 and more specifically list item 7.6 initialize and calculate the distances of each pair in the table. Then in element 7.7 the calculation of the 5 and 2 closest values of the three users being compared is done. Then in element 7.8 the Kemeny-optimal value is calculated and printed. The following pseudocode shows that the code is doing.

Algorithm 4 Step 1: Calculating the distances for each pair in the table

for $rating1 \in dist_table$ **do**
  for $rating2 \in dist_table[0]$ **do**
    for $item \in items$ **do**
      if $(index(rating1,'A') < index(rating1,'B') AND\ index(rating2,'A') < index(rating2,'B'))\ OR (index(rating1,'A') > index(rating1,'B') AND\ index(rating2,'A') > index(rating2,'B'))$ **then**
        $consistents++$
      end if
      if $(index(rating1,'A') < index(rating1,'B') AND\ index(rating2,'A') > index(rating2,'B'))\ OR (index(rating1,'A') > index(rating1,'B') AND\ index(rating2,'A') < index(rating2,'B'))$ **then**
        $inconsistents++$
      end if
      if $(index(rating1,'A') < index(rating1,'C') AND\ index(rating2,'A') < index(rating2,'C'))\ OR (index(rating1,'A') > index(rating1,'C') AND\ index(rating2,'A') > index(rating2,'C'))$ **then**
        $consistents++$
      end if
      if $(index(rating1,'A') < index(rating1,'C') AND\ index(rating2,'A') > index(rating2,'C'))\ OR (index(rating1,'A') > index(rating1,'C') AND\ index(rating2,'A') < index(rating2,'C'))$ **then**
        $inconsistents++$
      end if
      if $(index(rating1,'B') < index(rating1,'C') AND\ index(rating2,'B') < index(rating2,'C'))\ OR (index(rating1,'B') > index(rating1,'C') AND\ index(rating2,'B') > index(rating2,'C'))$ **then**
        $consistents++$
      end if
      if $(index(rating1,'B') < index(rating1,'C') AND\ index(rating2,'B') > index(rating2,'C'))\ OR (index(rating1,'B') > index(rating1,'C') AND\ index(rating2,'B') < index(rating2,'C'))$ **then**
        $inconsistents++$
      end if
    end for
    $dist[rating1][rating2] = consistents - inconsistents)/n$
  end for
end for

Algorithm 5 Step 2: Calculating the nearest 2 and 5

```
for element ∈ nearest_table do
    for row ∈ dist_table do
        for dist ∈ row do
            if element = dist then
                if dist > min(element) then
                    min(elemenet) = dist
                end if
            end if
        end for
    end for
end for
```

Algorithm 6 Step 3: Calculating the Kemeny-optimal

```
for element ∈ nearest_table do
    for value ∈ element do
        unique_elements = []
        if NOTvalue ∈ unique_elements then
            append(unique_elements, value)
            count++
            sum+ = value
        end if
    end for
end for
```

2. Correlation based on adjusted cosine

For example, suppose John, Jack and Kate want to buy a product together from the list of items in the example. First, to reduce the problem created by users evaluating either only positively or only negatively, we should normalize the values in the way we had shown in the example. So the table with the three users will become :

										
John		0.4		-0.6	0.4	0.4				-0.6
Jack	0	0	0				0		0	0
Kate		0.6	-0.4	-0.4		0.6		0.6	-0.4	-0.4

Then we need to find the similarity of users by pairs, so we need to calculate $Sim_{12}, Sim_{13}, Sim_{23}$
. For the similarity of John and Jack Sim_{12} will be :

$$M_1 = \frac{0.4 - 0.6 + 0.4 + 0.4 - 0.6}{5} = 0 \tag{2.36}$$

$$M_2 = 0 \tag{2.37}$$

$$M_3 = \frac{0.6 - 0.4 - 0.4 + 0.6 + 0.6 - 0.4 - 0.4}{7} = 0.2 \tag{2.38}$$

So the ratings predictions for user John will be :

(a) For the camera :

$$r_{1,camera} = M_1 + \frac{Sim_{1,2} * (r_2 - M_2) + Sim_{1,3} * (r_3 - M_3)}{Sim_{1,2} + Sim_{1,2}} =$$
$$= M_1 + \frac{Sim_{1,2} * (r_2 - M_2) + Sim_{1,3} * (r_3 - M_3)}{Sim_{1,2} + Sim_{1,2}}$$
$$\frac{\sum_{i \in I_x \cap I_y} (r_{xi} - \mu_x)(r_{yi} - \mu_y)}{\sqrt{\sum_{i \in I_x \cap I_y} (r_{xi} - \mu_x)^2} \sqrt{\sum_{i \in I_x \cap I_y} (r_{yi} - \mu_y)^2}} =$$
$$= 0 \tag{2.39}$$

And following the same procedure we can complete the Array :

										
John		0.4		-0.6	0.4	0.4				-0.6
Jack	0	0	0				0		0	0
Kate		0.6	-0.4	-0.4		0.6		0.6	-0.4	-0.4

2.1.2 Average functions

This method uses an average or median of user ratings and a weight value for each user which is used to avoid recommendations that would cause strong dissatisfaction for some team members. For example, a recommendation of a travel package that contains some kind of difficult sporting activity such as skiing or a walking trail would be quite unpleasant to a group containing a person with a physical disability.

2.1.3 Average functions with generalized liking

These functions use features of the two techniques mentioned earlier. After excluding negatively rated products, the mean or median of the group ratings is found. As this technique uses the average of those who showed the greatest liking for products, it is very insistent on the phenomenon of transmission of emotions, positive or negative.

2.2 Group tensions in knowledge-based systems

As we have seen, knowledge-based systems use user preferences to generate their recommendations as opposed to the aforementioned systems that use the ratings of one or many users. The process is almost the same, each user should describe their preferences and then the items that coincide with the preferences of most users in the group are recommended to the group. The phenomenon of active feedback offered by these systems, of group consensus before consuming the product makes these systems highly compatible with

group recommendations. For example, suppose John, Jack and Kate want to buy a product together from the list of items in the example. First, to reduce the problem created by users rating either only positively or only negatively we should normalize the prices in the way we had shown in the example . So the table with

Chapter 3

Deep Graph learning

As we have seen, collaborative filtering systems rely on historical data from the user and other users of the system to generate their recommendations. These methods due to their dependence on the evaluations made rather than the user's characteristics suffer from significant drawbacks such as the problem of cold start and scattered data. The cold start problem, as mentioned above, occurs when either the system is new, in which case there are not enough evaluations to produce a reliable recommendation, or when the user is new and users who are highly relevant to the user cannot be properly selected. We have seen that content-based and knowledge-based systems deal with this problem decently, but it is not always amenable to such a system. In this chapter we present a collaborative filtering method based on neural network graphs that uses machine learning elements to try to address this problem. In these methods, however, attention is paid to the characteristics of the products, neglecting the key factor of user characteristics.

3.1 Tensions based on enhanced two-sided knowledge graphs

Despite the positive results of this method there are some negative elements, some of which areYang and Cai, 1999:

1. Ignore user and product attributes
   Most recommendation systems based on collaborative filtering use only historical user data to generate their recommendations, ignoring important user and product attributes.

2. Ignoring importance of features

While some studies use user attributes, they ignore the importance of each attribute.

3. Insufficient research on user characteristics

4. There is little research on user preferences in which user preferences are generated by complex processes containing user attributes.

The extraction of user preferences, as well as the others mentioned above, are important elements for mitigating the cold start problem as well as the data scattering problem. To avoid these problems, a personal recommendation method using knowledge graphs is proposed.

3.1.1 Knowledge graphs

Knowledge graphs are data structures designed to model a group of objects, as nodes, and the relationships between them, as edges. As the use of graphs in domains such as machine learning enables a powerful representation, they are used in various scientific domains in various forms. Graphs focus on problems such as node classification, edge prediction, and clustering.One group of graphs, neural network graphs, have gained widespread popularity in many fields due to their good performance.
Graphs started to appear in the scientific community many years ago. In particular, the 1990s saw the launch of Recurrent graphs,Sperduti and Starita, 1997 and the 2000s saw the introduction of recurrent neural networks and feed forward neural networks,Scarselli et al., 2009Micheli and Alessio, 2009. Although successful, their materialization technique is to create state transitions in graphs and converge them through iterations, which limited their scalability and reproducibility. In recent years, breakthroughs in the field of deep neural networks, in particular convolutional Lecun et al., 1998, have led to a rethinking of knowledge graphs. Despite their advantage of problem solving, due to their features of local connections, shared weight and multiple bodies, coherent neural graphs exclusively use data normalized on the basis of Euclidean geometry such as two-dimensional mesh data and one-dimensional sequencesZhou et al., 2020. Thus, knowledge graphs evolved which are based on the cohesive graphs and which can compact data items.

3.1.2 Nets of convolutional graphing

Cohesive graph networks have been shown to be effective in processing graphical data structuresZhou et al., 2020 unlike traditional deep learning methods.

In this paper, we apply the cohesive graph networks for feature propagation based on the knowledge graph presented earlier in the knowledge graph subsection. As mentioned above, the cohesive graph networks are a deep learning model for graphical data structures with very strong representation features. For each node of the knowledge graph, the CGI (Collective Graph Network) encodes the information of the neighboring nodes into real-valued eigenvectors in an unsupervised manner. The processes of representing and propagating the entities of the OSS contains two main processes, propagation and merging. In the propagation process, the attributes of nodes are propagated iteratively to neighboring nodes along their link. In the process of aggregation, the attributes of neighbouring nodes are merged to produce the embedded target nodes. Each convolution means that the merging of the neighbouring nodes of the previous layer produces the embedding of the current node.

3.1.3 Selective neural networks on graphs

Cohesive neural networks offer an efficient architecture for the purpose of mining statistical patterns with high content with particular utility in large-scale and multidimensional data. Their key property is to learn local static structures and then synthesize them to generate multi-layered hierarchical patterns, which has led them to innovative successes in image, video and sound recognition.
More specifically, coherent neural networks manage to extract local static input data elements (data or signals) revealing local features. These local features are distributed among data domains and can be identified by local selective filters or kernels identified from the data set. A key feature of the selective filters is their tolerance to drift, which makes them spatially immutable. By local kernels we mean filters that extract local features independent of the volume of input data, with a much smaller size. These features make Colectic Neural Network Graphs a very useful and powerful tool on data classified as discontinuous or non-Euclidean such as social networking site data, text documents and telecommunication network history data. According to the article, despite the merits of coherent neural network graphs, even better results can be achieved by exploiting the characteristics of traditional spectral coherent neural network graphs and by offering more control over the support of the filters, thus achieving better results. In this paper Defferard et al., 2019 propose a generalization of the coherent neural networks from small-dimensional normalized lattices where information is represented in large irregular high-dimensional domains represented

by graphs. This generalization requires three steps :

1. Learning fast local spectral filters

2. Learning fast local spectral filters Two methods are mainly used to determine the coordinate filters, through spatial or spectral approaches, with the spectral one being predominant due to the weakness of the spatial one in matching local neighbours. The approach by spectral determination of coherent filters using

3. Graph hardening
   The process of graph hardening requires the creation of neighbors with significance where similar edges are clustered together. While there are several clustering methods most compatible for this particular case(Multidimensional clustering) seems to be the Glaclus**20** multilevel clustering method due to its efficiency on a large number of graphs. In this method, each level produces a graph representing the data domain through a different analysis.
   Glaclus' greedy algorithm is based on the simple technique of Metis**21**, where for each scaling plane a is calculated by selecting an intersection j and matching it with an unselected neighbour j. These two edges are selected and their joint weight is calculated:

$$W_{(i,j)} = 1/d_1 + 1/d_2 \tag{3.1}$$

4. Fast concentration of the signals of the graph
   Finally, after hardening to make it easier to cluster the graph, the vertices can be arranged in a certain way. The teaching process is done in two steps, creating a binary graph and reorganizing the vertices. Normal edges have either 2 children of normal edges or an isolated edge and a false edge. Finally, each false edge has two children of false edges.

3.1.4 Syntactic graph networks based on heterogeneous user mobility graphs

Synlectic graph networks were exploited in another cat article in which it was used to create for the first time a model that uses social relationships from a volume of user movement data to derive a heterogeneous graph. This graph consists of a user encounter graph, a social graph, a bilateral user and location graph, and a location intersection graphSchlichtkrull et al., 2018.

This method consists of the following four steps:

1. Create the heterogeneous user mobility graph.

2. To give features to the user mobility model and to create a model of relationship propagation, a heterogeneous user mobility graph that represents the relationship between users, between user and location and between locations using different kinds of information is proposed. This graph is divided into three parts, the user layer, the location layer and the user and location feedback layer as shown in Figure 5.2. The figure is divided into three levels, the user level, the user-location level and the location level. Each edge of the graph in the user plane represents the mobility relationship between the two users while the weight is determined according to their encounter frequency. In the user-location layer, the weight of each user-location relationship determines the user's visit rate to that location. Finally, at the third level, each edge identifies how often the two sites are visited serially, with the weight indicating how often these two sites are visited serially and the edges indicating the most popular sequences.

3. Unsupervised node representation learning. In this step we make an extension of the use of Neural Network GraphsSchlichtkrull et al., 2018 for semi-supervised classification of entities so that it can be used as a coder for existing models for predicting relationships in knowledge graphs with the ultimate goal of unsupervised learning node representation in heterogeneous graphs. These graphs use the following formula :

$$h_i^{(i+1)} = \sigma(\Sigma_{r \in R_v} \Sigma_{j \in N(i)} \frac{1}{C_{i,r}} W_r^{(}l) h_j{}^{(l)} + W_0^{(l)} h_i^{(l)} + b) \tag{3.2}$$

   Which represents the rule for propagation expressed in the message passing architecture, for gathering information from neighboring nodes and sending this information to the next layer. The loss for a particular node can be calculated by the formula:

$$L_(u) = -\log(\sigma(\phi_u \phi_v)) - \lambda \log(\sigma(\phi_u \phi_w)) - \lambda_n eg \sum z \in (NEG(u)) \log(-\phi_u \phi_z) \tag{3.3}$$

4. Extraction of node characteristics from the trajectories
   To extract node attributes in this step, the user mobility recognition method using trajectory embeddings **19** is used by adapting it to learn

user and location embeddings. Specifically, in the first step the user trajectory is divided into trajectory segments for which location embeddings are learned separately using a content probability prediction maximization model. Then, the sub-segments of the location embedding trajectories are used as parameters in a recurrent neural network. To learn this network, the trajectory subsegments are mapped to users via a fully integrated layer.

5. Semi-supervised node representation learning
   Finally, to enhance the performance of the method if a bit of friendly relations are known. By exploiting the information of neighboring nodes as local content in the loss function, the calculation of the semi-predictable loss constant $(L_semi) = -\log((\phi_u)(\phi)z)$ can be calculated as follows:

$$L = L_{unsup} + (\lambda)_{semi} L_{semi} \tag{3.4}$$

The results showed better performance on the three datasets used than the other methods. A key factor is that in addition to using the extrinsic characteristics of the nodes, this method can also exploit the structure of heterogeneous graphs to capture the multidimensional relationships between nodes.

3.1.5 Blind neural networks

3.1.6 Personalized online learning recommendation

An analysis of the framework of this method is given below. The figure shows the network architecture, which is divided into two parts. On the left, the unit of knowledge graph based reinforcement is shown, while on the right, the unit of recommendation is represented. The module of knowledge graph base reinforcement consists of two parts, the embedded learning graph base propagation and the embedded user graph base propagation. In the right section, we firstly represent the exploitation of the attention network so as to generate embedded user preferences for learning interaction, and then, taking as parameters the latent embedding of the user and the latent embedding of the course, it generates through a deep neural network the predicted probability.

The process of fission and merging can be represented as follows:

.

$$q_i^{(i+1)} = \sigma(\Sigma_{r \in R_v} \Sigma_{j \in N(i)} \frac{1}{C_{i,r}} W_r^l q_j^{(l)} + W_0^{(l)} q_i^{(l)} + b) \tag{3.5}$$

Where $(q_i)^(l) \in R^(D^l)$ represents the latent embedding of node q_i in the l level network. $W_r^(l)$ is the weight matrix b is the bias index
$N(i)$ is the set of neighbouring nodes of this node with correlation r
$c_(i,r) = |N(i)|$ represents the number of neighbours of the node and σ is the activation function.

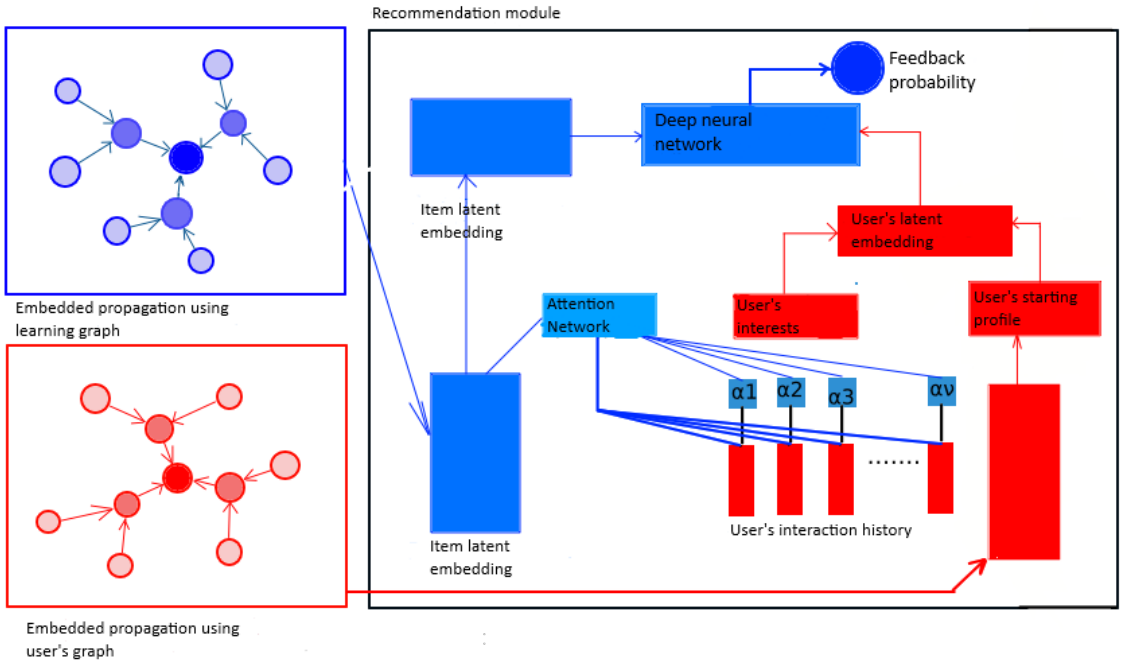


Figure 3.1.1: Customized online learning recommendation

3.1.6.1 Example

In this section of the chapter, an example will be given to understand the process of computing the propagation and computing the recommendation. In this example we will assume three atoms, Peter, Mary and Nick for which through their interactions we will try to predict the characteristics of Peter taking into account his interactions with the others and the characteristics of the other atoms.

As shown in Figure 5.2 these individuals besides their characteristics (Occupation, Research Interests) there are also some alleles with other research articles as well as some relationships between them. The purpose is to predict Peter's occupation and interests.

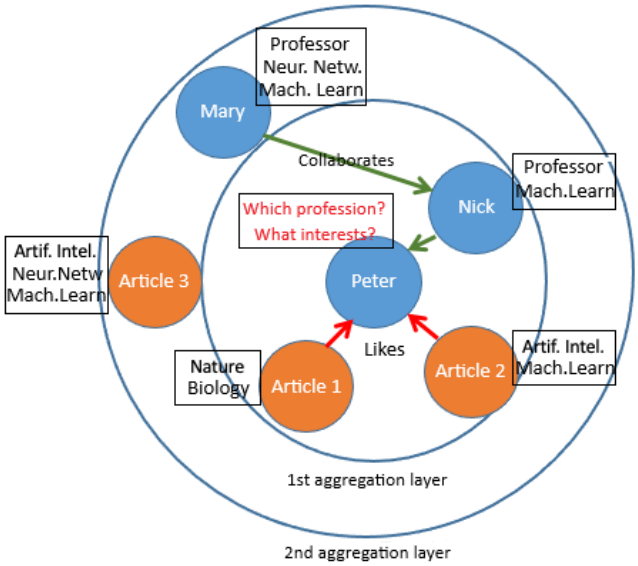


Figure 3.1.2: Deep machine learning example

Figure 3.1.3 shows the heterogeneous user mobility diagram from another perspective by adding the interactions that occur by merging Mary's and Nick's interests with the articles.

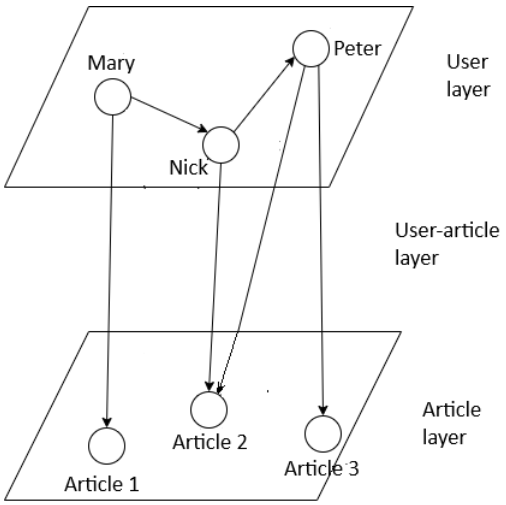


Figure 3.1.3: User Mobility Graph

As mentioned above once the graph is created then for each layer the latent embedding index is calculated and propagated to the next layer. So first the weights of the nodes should be calculated. Considering that the

weight of an edge indicates the frequency of cooperation between the two users. At the user-article level as mentioned above is the frequency of the user's feedback with that particular article. So, you translate by whether the user likes the article or if it coincides with his/her likes. At the article level, the weight of each edge means the commonalities of the two vertices and the weight of each edge represents the frequency of visitation serially which you translate for this example to the commonalities of the articles.
Thus, the edge weights tables for the user layer can be created. For the user level each edge represents the cooperation of the two users for the semi-supervised model. We can additionally compute an edge that represents the frequency of the preferences of the two users, i.e. how many shared likes they have. Thus, by calculating the weight as the number of shared preferences (articles) the values can be calculated:

$$W_{(M,N)} = 2, W_{(N,P)} = 1, W_{(P,M)} = 1 \tag{3.6}$$

Then, we can proceed with the calculation of the weights at the user-article level. Each edge denotes the user's liking of that article while the weight of the edge denotes how many people like that article and more specifically the frequency of liking the article.
So we can calculate for Mary:

$$W_{(M,art1)} = 0, W_{(M,art2)} = 0, W_{(M,art3)} = 1, \tag{3.7}$$

For Nick:

$$W_{(N,art1)} = 0, W_{(N,art2)} = 2, W_{(N,art3)} = 0, \tag{3.8}$$

And for Peter:

$$W_{(P,art1)} = 1, W_{(P,art2)} = 2, W_{(P,art3)} = 0 \tag{3.9}$$

For the article level each edge shows the sequence of the two articles and the weight shows the frequency of the sequence, i.e. how often users like an article when they like it and the one you associate with the edge. So, for articles 3 and 2:

$$W_{(art3,art2)} = 2 \tag{3.10}$$

For articles 2 and 1:

$$W_{(art2,art1)} = 2 \tag{3.11}$$

While for articles 1 and 3 the weight is 0 since for no user this sequence exists (to like both articles). This process of weight calculation can be easily calculated using the Python programming language as shown in appendix 7.11.
Next, Gilmer's method **21** will be used to perform the propagation and merging step with the help of the formula mentioned above 3.2. In this step, information is collected from the neighbors of a node and this information is propagated to the next level.
Practically, this means that we will use formula 3.2 by calculating for each relation in level l the weights of each neighbor with the final goal of articleizing them and calculating information for level l+1. So for our example we need to compute for the two nodes in the second layer the propagation information to propagate to the first layer. For the activation function we will use the ReLU(a) (Rectified linear unit) function which is the preferred method, since the sigma function proved to be a very slow method of learning the graph. This function is nothing but a function that chooses the maximum between 0 and the value you give it. Practical result of this function is zero if the parameter is less than zero and the parameter if the value is greater than zero as shown in the formula:

$$ReLU(a) = max(0, a) \tag{3.12}$$

Thus, for the representation of the Peter's node under consideration we will have:

$$h_i^{l+1} = max(0, \sum_{r \in R} \sum_{jin(N_i)^r} \frac{1}{C_i, r}(W_r)^l (h_j)^l) \tag{3.13}$$

where C_i, r is the number of neighbours of node i with respect to the directed edges of type r, w_r is the matrix of weights in the l-plane. Peter's preference representation process can be done through a neural network in two main ways, either based on his article interaction data or his collaborations with other users. Specifically, for a node under consideration, its latent vector should be calculated, which can be represented as follows:

$$h_i = \sigma(WAGGREGATE(q_j, \forall j \in C(i))) = \max 0, WAGGREGATE(q_j, \forall j \in C(i)) \tag{3.14}$$

with AGGREGATE to represent by a cumulative articleization function, usually average, for the characteristics of the neighboring nodes of the node

under consideration. Alternatively, the CONCATENATION operator can be used, which concatenates the vectors of the articulated vectors.
Thus, using the average, formula 5.14 will become:

$$h_i = \max(0, W \sum_{j \in C(i)} \frac{1}{C(i)} q_j) \tag{3.15}$$

which must be calculated for each edge going to Peter's node on the second level, three times for the nodes of Nicks, article 1 and article 2.
The calculation of the average is done for each type of edge separately, i.e. it will be calculated for Peter's collaborations separately from his article likes.
Thus, for the aggregation of Nick's information will apply:

$$h_{Nicks} = \max(0, W \sum_{limits_{j \in C(i)} \frac{1}{C(i)} q_j}) \tag{3.16}$$

Where W is the weight matrix calculated previously:

1	0	0
0	2	0
1	2	0

And specifically for Peter:

Where for the columns we have articles 1, 2 and 3. The information will be

1	2	0

passed on to Peter with the help of formula 5.13. So for Nick

$$h_{Nicks} = \max(0, q_j) = 0, 2, 0 \tag{3.17}$$

Which represents the information Teacher, Neural Networks, Learning Engineering And for Nick's information transmitted to Peter, since he is the only one transmitting information to Peter, his information will be aggregated with the information sent by Mary. So it will be valid

$$h_{maria} = \max(0, q_j) = 0, 0, 1 \tag{3.18}$$

representing the information Teacher, Neural Networks, Learning Engineering Thus giving the prediction that Peter will be a professor with an interest in Machine Learning. For articles it will apply:

$$h_{art1} = \max(0, W_t imes q_j) = W1, 1, 0, 0 \tag{3.19}$$

$$h_{art2} = \max(0, W_t imes q_j) = W \cdot [0, 0, 1, 1] \tag{3.20}$$

Where scaled and multiplied by their weights will be:

$$h_{art2} = \max(0, W \times q_j) = 0.25, 0.25, 0.75, 0.75 \tag{3.21}$$

Which will be combined with the information from Nick giving the final result.

Algorithm 7 Step 1: Calculating the weights

```
for coop ∈ cooperations do
    for interest1 ∈ getInterests(coop[0]) do
        for interest2 ∈ getInterests(coop[1]) do
            if interest1 = interest2 then
                interests_count = interests_count + 1
            end if
        end for
    end for
end for
```

Algorithm 8 Step 2: Calculating the vector of person

```
for edge ∈ edges do
    if edge[0] = person then
        person(neightbors) = edge[1]
    end if
end for
h_i = getAttributes(person)
for node ∈ nodes do
    for node ∈ nodes do
        if node ∈ person(neighbors) then
            q_j = all_attributes(node)[1])
            if interest1 = interest2 then
                interests_count = interests_count + 1
            end if
        end if
    end for
    features = features + q_i
end for
product = features · interest_count
```

Chapter 4

Fairness and Bias

4.1 Bias

Many of the algorithms for the models mentioned suffer from the problem of bias. By bias we mean the tendency of an algorithm not to make recommendations impartially and objectively and to tend towards recommending goods that are either more popular or have been positively evaluated by the user as well as other items.
These elements will be discussed below, as well as some solutions to address them.

4.1.1 Popularity Bias

Many recommendation systems suffer from popularity bias, i.e. recommending products that are more popular thus avoiding products that are more rare and specialized. Recommendations of rarer and specialized products are essential for firms as these products are harder to discover by customers since there is not the time available for extensive catalogue searches.
Recommendation systems that select their recommendations according to the popularity of a product hold the advantage of better success. However, it has been shown that recommending based on other elements than the popularity of an item (whether the product is niche and whether it fully covers the catalogue) has a positive influence on the overall quality of the recommendation system. Beyond that, there are many users who prefer products of less popularity, who are repelled by a system that does not suggest such. And as a reference to question 1 about 1/3 of users have about 20 % unpopular artists in their profiles. Also, you infer that users with small profiles have

more unpopular artists than users with large profiles.
Finally, you give a way to study the popularity bias for different user groups. More specifically, you use the Group Average Popularity (GAP) technique proposed in the article Abdollahpouri and Masoud Mansoury, 2019 in which you propose the following formula:

$$\delta GAP = \frac{GAP(g)_r - GAP(g)_p}{GAP(g)_p} \tag{4.1}$$

where $GAP(g)_p$ measures the average popularity of the artist under the user profile p of a given group g and $GAP(g)_p$ is the average popularity of the artists promoted by an algorithm r to a group of users p. Specifically, the average popularity of the artist GAP(g) is defined as.

$$GAP(g) = \frac{\Sigma u \in g \frac{\Sigma i \in p_u \Phi(i)}{|p_u|}}{|g|}) \tag{4.2}$$

Still, publicity bias causes so-called filtering bubbles, i.e. restricting the user to a virtual bubble by suggesting certain products and preventing him from accessing others, making the user lose interest as well as other elements beyond aptitude. Thus, limiting the element of publicity bias can yield a balance between the benefits of favorability and other elements, with the ultimate goal of creating a better recommendation system.

4.1.1.1 Example

We will use formula 6.1 to find the amount of publicity bias that the group recommendation algorithm suffers from. Specifically, we will use the movielens100k data consisting of 943 users, 1682 movies, and 100000 ratings.‖ First, we make a categorization of users according to the percentage of popular movies in their profile. The thousand users will be divided into 3 categories, the blockbuster which is composed of the 20% of users with the highest percentage of popular movies in their profile, the niche which is the 20% with the least percentage of popular movies in their profile and the diverse which is composed of the remaining 60%. Figure 6.1 shows the categorization of users.

Using code 7.11 it can be calculated how much the group recommendation algorithm suffers from group bias. Still, for easier comparison and understanding of the algorithm we will compare it with two other recommendation algorithms, the random selection algorithm (Random) and the

most popular algorithm (MPIR). We first take the average group popularity values as shown in Figure 6.2.

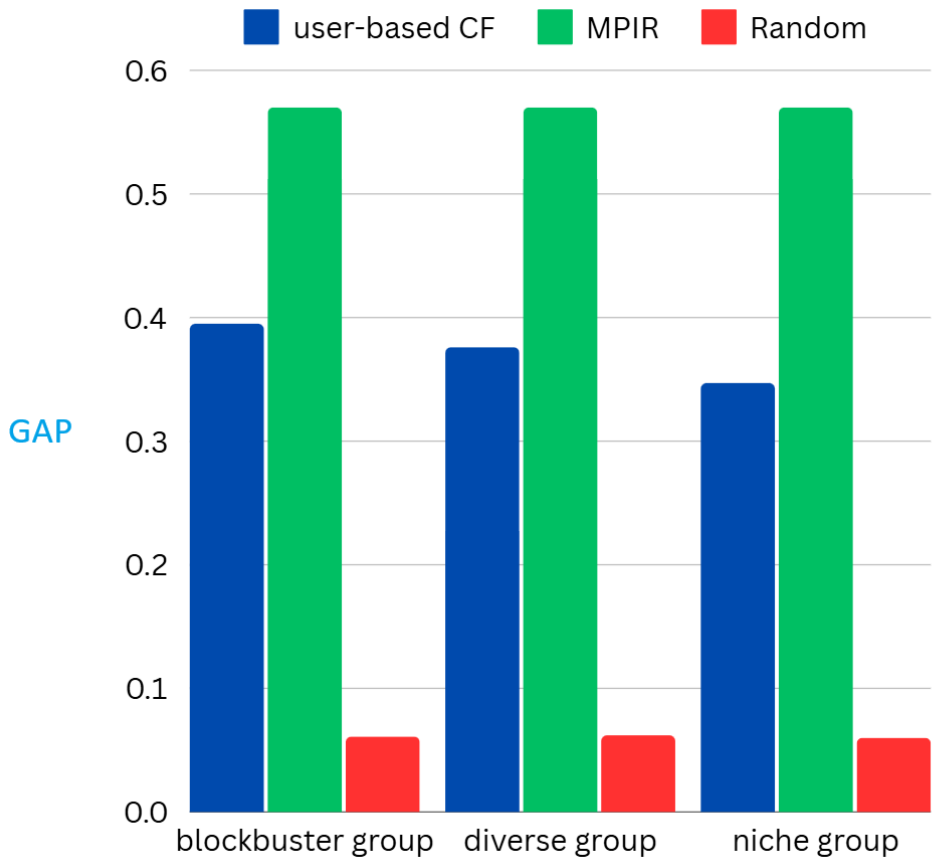


Figure 4.1.1: Graph of average publicity

So, we can create from the values we got the diagram 6.3.

The graph shows that the MPIR and user-CF algorithms suffer from popularity bias, which is obvious since MPIR recommends the most popular items while user-CF uses other users' recommendations to generate its recommendations, making it prone to recommend popular ones. One successful method to reduce the popularity bias is to reduce the k-nearest neighbors used to generate recommendations for the user under consideration in collaborative filtering. More specifically, the algorithm after subtraction of all associations of users with each other:

Computes the location of the k-nearest neighbors and stores it in a table as shown in the pseudocode below: After calculating the similarities for each user with each user as shown in the pseudo-code of step 1, we can go on to

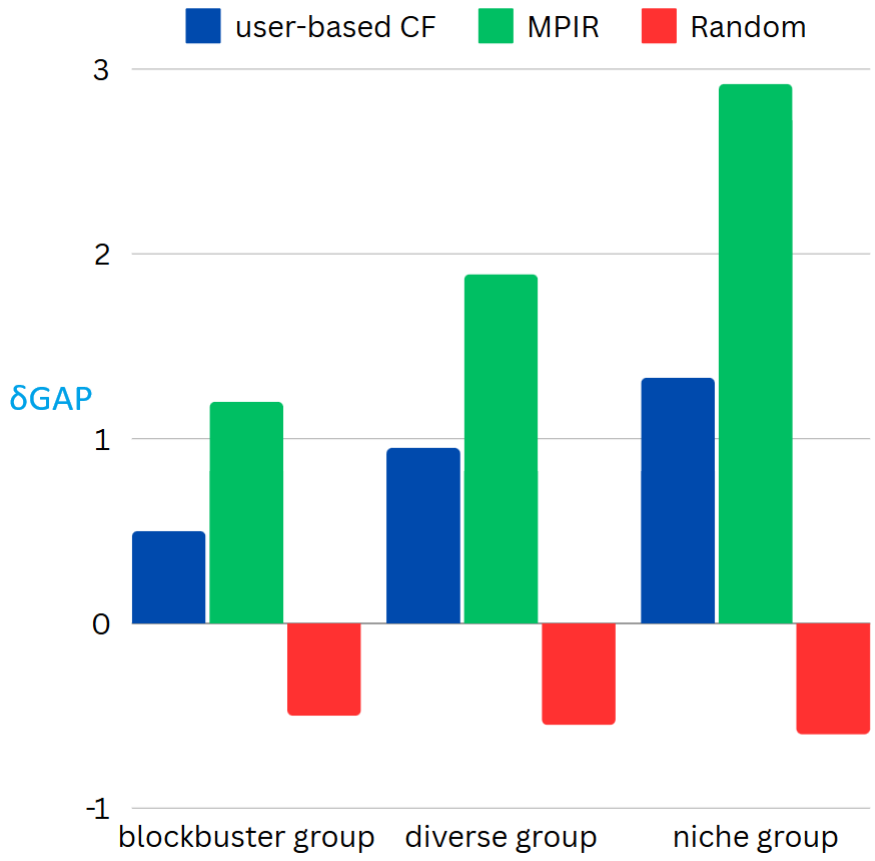


Figure 4.1.2: Publicity bias graph of different algorithms

calculate the highest k-nearest neighbors similarities and store their location in a table as shown in step 2. Then according to the recommendations of the users with the highest similarities we can calculate for each user the recommendations they will have as finally shown in step 3

Algorithm 9 Step 1: Calculating user similarities

for $user1 \in users$ **do**
  for $user2 \in users$ **do**
    for $item \in items$ **do**
      $nominator = nominator + rating(user1, item)$ - $mean\_value(user1)$ * ($rating(user2,item)$ - $mean\_value(user2)$)

      $denominator1 = denominator1 + (rating(user1, item)$ - $mean\_value(user1))^2$
      $denominator2 = denominator2 + (rating(user2, item)$ - $mean\_value(user2))^2$
      $recommendation = nominator$ / $square\_root(denominator1)$*$square\_root(denominator2)$
    end for
  end for
end for

Algorithm 10 Step 2: Finding the top n neighbors

for `recommendation` $\in$ `user_recommendations` **do**
  if $recommendation > minimum(user \in top_k group\_similarities$ **then**
    $top\_k - group\_similarities(user)(minimum(top\_k - group\_similarities)) = rating$
  end if
end for

Algorithm 11 Step 3: Finding k-number of top neighbors

for $user1 \in users$ **do**
  for `rating` $\in$ `user_rating` **do**
    if $recommendation \in top\_k - group\_similarities(user)$ **then**
      $user\_recommendation(user)(item) = list\_recommendations = calculate\_recomendation(user, item)$
    end if
  end for
end for

Algorithm 12 Step 4: Random recommendations

for $rec \in rand_recommendations$ **do**
    $rec = random(0, len(items))$
end for

Algorithm 13 Step 5: MPIR recommendations

$popular_items = sorted(items)[:3]$

We ran the same code 7.11 with k-nearest neighbour sizes of 10, 20 and 30, getting the results of Figure 6.4

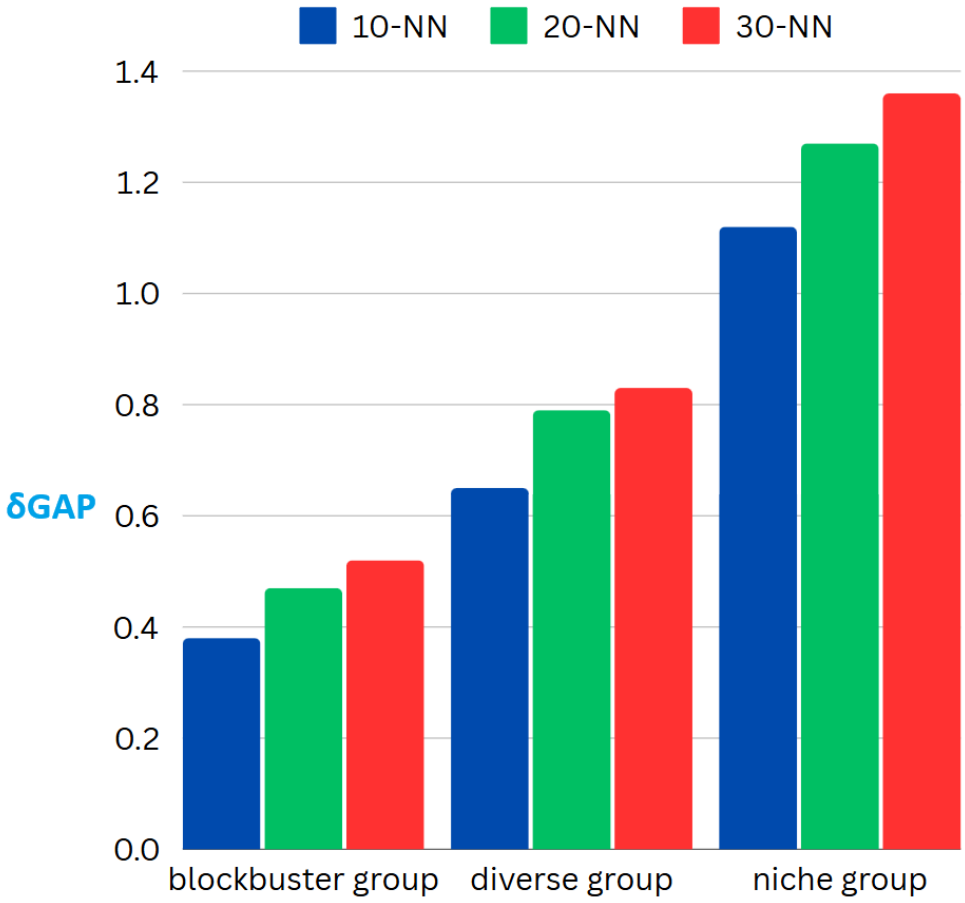


Figure 4.1.3: Publicity bias graph using different sized groups

It can be seen from the graph that by reducing the number of users, the publicity bias can also be reduced, which also makes the algorithm less

Algorithm 14 Step 6: Calculating the DGAP values

```
for group ∈ groups do
    for user_ratings ∈ l_user_items_prediction do
        for ratings ∈ user_ratings do
            user_ratings(nominator) += items(ratings)(popularity)
            p1+ = 1
            if rec ∈ l_max_prediction then
                user_ratings(nominator_cf) = user_ratings(nominator_cf + items(ratings)(popularity)
                p2+ = 1
            end if
            if rec ∈ rand_recommendations then
                user_ratings(nominator_random) = user_ratings(nominator_random) + items(ratings)(popularity)
                p3+ = 1
            end if
            if rec ∈ popular_recommendations then
                user_ratings(nominator_popular) = user_ratings(nominator_popular) + items(ratings)(popularity)
                p4+ = 1
            end if
        end for
        GAP_group = nominator/p1
        GAP_group_cf = nominator_cf/p2
        GAP_group_random = nominator_random/p3
        GAP_group_popular = nominator_popular/p4
    end for

    for group ∈ groups do
        group(DGAP) = group(GAP_group_cf) - group(GAP_group) / group(GAP_group)
    end for
```

successful.

Chapter 5

ResultsConclusion

In general, we used 3 basic recommendation techniques to observe the recommendations they give and how to take advantage of each.
We managed to provide some examples, investigate them and run into some conclusions. More precisely, we checked the group recommendations using the Kemendy-optimal using the Pierson corellation. Then for merging the ratings we saw how the Kendall tau method is used. We provided examples and the Python code for this. Then, we run into a conclusion in chapter 5 that whis method suffers from the popularity bias by comparing the popularity bias using the DGAP method. We also provided the Python code which can calculate these numbers for different user groups
We also managed to see and investigate some results in deep machine learning method using a heterogeneous graph and provide the algorith using Python for that

References

Abdollahpouri, Himan and Bamshad Mobasher Masoud Mansoury Robin Burke (2019). "The Unfairness of Popularity Bias in Recommendation". In: *The Unfairness of Popularity Bias in Recommendation.*

Aggarwal, Charu C. (2016). *Recommender systems, The textbook.* Vol. 1. Springer.

Defferard, Michael, Xavier Bresson, and Pierre Vandergheynst (2019). "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *EPFL, Lausanne, Switzerland.*

Lecun, Y. et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: 10.1109/5.726791.

Micheli and Alessio (2009). "Neural Network for Graphs: A Contextual Constructive Approach". In: *IEEE Transactions on Neural Networks* 20.3, pp. 498–511. DOI: 10.1109/TNN.2008.2010350.

Scarselli, Franco et al. (2009). "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1, pp. 61–80. DOI: 10.1109/TNN.2008.2005605.

Schlichtkrull, Michael et al. (2018). "Modeling relational data with graph convolutional networks". In: *In European Semantic Web Conference (ESWC).*

Sperduti, A. and A. Starita (1997). "Supervised neural networks for the classification of structures". In: *IEEE Transactions on Neural Networks* 8.3, pp. 714–735. DOI: 10.1109/72.572108.

Yang, Shuang and Xuesong Cai (1999). "Bilateral Knowledge Graph Enchanced Online Course Recommendation". In: *Bilateral Knowledge Graph Enchanced Online Course Recommendation.*

Zafarani, Reza (2014). "Social media mining, an introduction". In: *Social media mining, an introduction.* URL: http://dmml.asu.edu/smm.

Zhou, Jie et al. (2020). *AI Open.* Department of Computer Science et al.

Chapter 6

Appendices

6.1 Programming code

This section presents the code needed to implement this project. It should be emphasized that the implementation was done in Jupiter notebook, a programming code implementation platform that can use server resources to execute the code, thus enabling implementation without installing the specific programming language.

6.1.1 Rating prediction

This piece of code represents the process of predicting the empty ratings of a set of users in a set of ratings is done

6.1.1.1 Array initialization and mean value calculation

In the following piece of code the initialization of the ratings table is done and then the calculation of the average of each user. Highlight some specific points in the code: A table(mo) is created which will contain the averages of each user. For each user in the ratings table, after checking if the sum of his ratings is not 0 to avoid division by 0, put his average in the appropriate place.

Listing 6.1: Calculating user average

```
import math

ratings = [[0,2,0,1,2,2,0,0,0,1],          #John
           [1,2,2,0,0,2,1,1,0,0],          #Mary
```

```
                    [1,1,1,0,0,2,1,0,1,1],      #George
                    [0,0,2,0,0,1,0,0,0,0],      #Nicole
                    [2,0,0,2,0,2,2,2,1,0],      #Geo
                    [0,1,0,0,2,0,0,1,0,2],      #Xristos
                    [1,0,1,0,1,0,2,0,2,0],      #Peter
                    [0,2,1,1,0,2,0,2,1,1]]      #Katerina
mo = []
#for every rating in line count the number of ratings
#and the number of items
#and then calculate the mean value
for users in ratings:
    numberOfItems=0
    for product in users:
        if product != 0:
                numberOfItems+=1
    if numberOfItems!=0:
        mo.append(sum(users)/numberOfItems)
    else:
        mo.append(0)
```

6.1.1.2 Calculations of similarities between users

In the following code fragment the calculation of the similarities of each user is done. First, a table is created containing the similarities of users(Number of users * Number of users). Then in an iteration that goes through each cell of the table it sees if the given object has been evaluated by both users and if so then it puts the value of the type in the three variables. The three variables calculates are. var1: one user's rating minus his average over the other user's rating minus his average
var2: the rating of the user minus his ratings' mean value
var3: other users rating minus his ratings' mean value
And then after checking if the denominator is not zero, it calculates the first one by the product of the square roots of the others.

Listing 6.2: Calculating similarities between users

```
similarities = [[0 for x in range(len(ratings))] for y
    in range(len(ratings))]          #pearson similarity
    array
#for every pair of users
#calculate the 3 variables that make the pearson
    similarity
```

```
#and then append it to similarities table
#eventually a 2d array will be created which will show
   for
#every user their similarity with every user
#var1 numerator
#root(var2)*root(var3) denumerator
counter = 1
for user in ratings:          #loop for users
  for user2 in ratings[counter:]:
    var1,var2,var3 = 0,0,0
    for item in range(0,len(ratings[0])) :
      if (user[item]!=0 and user2[item]!=0):
        var1 += (user[item]-mo[(ratings.index(user))])
           *(user2[item]-mo[(ratings.index(user2))])
        var2 += ((user[item]-mo[(ratings.index(user))])
           **2)
        var3 += ((user2[item]-mo[(ratings.index(user2))
           ])**2)
      if  var2!=0 and var3!=0 :
        similarities[ratings.index(user)][ratings.index
           (user2)]=round(var1/(math.sqrt(var2)*math.
           sqrt(var3)),2)
    counter+=1
for user in similarities:
    print(user)
    print('\n')
```

6.1.1.3 Calculation of similarities' max values

The following code fragment calculates the three maximum similarity values of each user. First, it creates two tables of length equal to the number of users and then initialize one with the three largest similarity values of each user and the other with the user who holds each of these three values

Listing 6.3: Maximum similarities selection

```
for user in range(len(similarities)):
    for rating in range(len(similarities[user])):
        similarities[rating][user]=similarities[user][
           rating]

for user in similarities:
```

```
        print(user)
        print()
print(mo)
#The array with the 3 max values for each users
maxs=[]
#the array with the users' three biggest similarities
maxu=[]
for user in similarities:
    maxs.append([-1,-0.99,-0.98])
    maxu.append([0,0,0])
for user in similarities:
    for sim in user:
        if sim > min(maxs[similarities.index(user)]):
                maxs[similarities.index(user)][maxs[
                    similarities.index(user)].index(min(
                    maxs[similarities.index(user)]))]=
                    sim
                maxu[similarities.index(user)][maxu[
                    similarities.index(user)].index(min(
                    maxu[similarities.index(user)]))]=
                    user.index(sim)
for user in range(len(maxs)):
    maxu[user][0]=similarities[user].index(maxs[user
        ][0])
    maxu[user][1]=similarities[user].index(maxs[user
        ][1])
    maxu[user][2]=similarities[user].index(maxs[user
        ][2])
for i in maxs:
    print(i)
for i in maxu:
    print(i)
```

6.1.1.4 Predicting user ratings

In the following piece of code each empty rating of each user is predicted. In one iteration that goes through each cell of the ratings table it checks if that particular rating is empty and if it is, it looks at the three users with the highest similarity calculated previously. Then, for the users who have ratings for that product it calculates the value of the similarity times the difference of their rating from its average and adds it to the predictions variable. It

then adds its similarity to the variable 'simis'. Finally, after checking for a non-zero denominator, it calculates the product of its mean and the quotient of predictions. divided by 'simis'.

Listing 6.4: Rating prediction

```
predictedRatings =ratings
for user in ratings:
    counter=0
    for rating in user:
        prediction=0
        simis=0
        if rating==0:
            if ratings[maxu[ratings.index(user)][0]][
               counter] !=0 :
                prediction+=similarities[ratings.index(
                   user) ][ maxu[ ratings.index(user)
                   ][0]] * (ratings[ maxu[ ratings.
                   index(user) ][0] ][ counter ]-mo[
                   maxu[ ratings.index(user) ][0] ] )
                simis += similarities[ ratings.index(
                   user) ][ ( maxu[ ratings.index(user)
                    ][0] ) ]
            if ratings[maxu[ratings.index(user)][1]][
               counter] !=0 :
                prediction+=similarities[ratings.index(
                   user) ][ maxu[ ratings.index(user)
                   ][1]] * (ratings[ maxu[ ratings.
                   index(user) ][1] ][ counter ]-mo[
                   maxu[ ratings.index(user) ][1] ] )
                simis += similarities[ ratings.index(
                   user) ][ ( maxu[ ratings.index(user)
                    ][1] ) ]
            if ratings[maxu[ratings.index(user)][2]][
               counter] !=0 :
                prediction+=similarities[ratings.index(
                   user) ][ maxu[ ratings.index(user)
                   ][2]] * (ratings[ maxu[ ratings.
                   index(user) ][2] ][ counter ]-mo[
                   maxu[ ratings.index(user) ][2] ] )
```

```
                simis += similarities[ ratings.index(
                    user) ][ ( maxu[ ratings.index(user)
                     ][2] ) ]
            if simis!=0:
                predictedRatings[ ratings.index(user)
                    ][counter]=round(mo[ratings.index(
                    user)] + (prediction/simis),2)
            else:
                predictedRatings[ ratings.index(user)
                    ][counter]=mo[ratings.index(user)]
        counter+=1
for i in predictedRatings:
        print(i)
        print()
```

6.1.1.5 Calculate Kendall tau distance between a group of users

The following code fragment calculates the distance between a group of users, specifically the group of users 0,2,7. It first creates a table of the users in the group and initializes it with the ratings of the users in the group. Then a sorting of this table is done according to the first user in the list, user 0. Finally the calculation of the Kendall distance tau is done. It compares each rating of both users with all the subsequent ones and if they agree then it increases the variable c. If they are inconsistent then it increases the variable d and in each case it increases the variable n. At the end of comparing each evaluation with the others it calculates the distance using the formula f = (c-d)/n. It should be emphasized here that consistency means that both (one user's and the other user's) are either smaller or larger than the one he is comparing. If one is shorter and the other is longer then the values are considered inconsistent. If either is equal(r1=r2) then consider neither consistent nor inconsistent.

Listing 6.5: Calculating Kendall tau distance

```
groupR = []
groupM1=0
groupM2=1
groupM3=7
for user in predictedRatings:
    if predictedRatings.index(user)==groupM1 or
       predictedRatings.index(user)==groupM2 or
       predictedRatings.index(user)==groupM3:
```

```
        groupR.append(user)
print(groupR)
sortRatings = groupR
for rating in range(0,len(groupR[0])-1):
    for ratingR in range(rating+1,len(groupR[0])):
        if groupR[0][rating]>groupR[0][ratingR]:
                tmp = groupR[0][ratingR]
                groupR[0][ratingR]=groupR[0][rating]
                groupR[0][rating]=tmp
                tmp = groupR[1][ratingR]
                groupR[1][ratingR]=groupR[1][rating]
                groupR[1][rating]=tmp
                tmp = groupR[2][ratingR]
                groupR[2][ratingR]=groupR[2][rating]
                groupR[2][rating]=tmp

print(groupR)
c = 0
d = 0
n = 0
user1=1
user2=2
for rating1 in range(0,len(groupR[user1])):
    for rating2 in range(rating1+1,len(groupR[user1])):
        if ((groupR[user1][rating1]<groupR[user1][
           rating2] and groupR[user2][rating1]<groupR[
           user2][rating2]) or
                        (groupR[user1][rating1]>groupR[
                           user1][rating2] and groupR[
                           user2][rating1]>groupR[user2
                           ][rating2] )):
                c+=1
        elif ((groupR[user1][rating1]<groupR[user1][
           rating2] and groupR[user2][rating1]>groupR[
           user2][rating2]) or
                        (groupR[user1][rating1]>groupR[
                           user1][rating2] and groupR[
                           user2][rating1]<groupR[user2
                           ][rating2])):
                d+=1
        n+=1
```

```
print(f"c= {c} d= {d} n= {n}")
print(f"t= {(c-d)/n}")
```

6.1.2 Rating merging based on Kemeny-optimal

In this section we merge ratings in a table of three products from three users.

6.1.2.1 Table arithmetic and distance calculation

In this subsection, initialize the table of ratings of three users and compute the distances of all possible pairs. At the beginning, initialize a table that has all possible pairs in the first row and first column. The aim is to initialize the empty cells with the distance of each combination of ratings to the corresponding one. Also,

Listing 6.6: Array initialization and distance calculating

```
# Here, we complete the array#
#For every array pair we look at the the
    alphabetiacally sorted rate pair, the sum(AB, AC, BC
    )
#and if its index in one element is either bigger or
    smaller they are compatible.
#(index_it_first(A) < index_in_first(B) AND
    index_in_secont(A) < index_in_second(B))
#(index_it_first(A) > index_in_first(B) AND
    index_in_secont(A) > index_in_second(B))
#if they are different they are considered incompatible
#(index_it_first(A) < index_in_first(B) AND
    index_in_secont(A) > index_in_second(B))
#(index_it_first(A) > index_in_first(B) AND
    index_in_secont(A) < index_in_second(B))
#if the indexes are the same then they are not
    considered anything
#Then we use the function (c-d)/n where c is the number
     of compatible pairs, d the incompatibles and n the
    pairs
#and we save the value in the array
ratings=[['A','B','C'],
         ['B','C','A'],
         ['B','A','C']]
rats=['A','B','C']
#the distance array
distT = [["a/a","ABC", "ACB", "BAC", "BCA", "CAB",
    "CBA"],
         ["ABC",0,0,0,0,0,0],
```

```
        ["ACB",0,0,0,0,0,0],
        ["BAC",0,0,0,0,0,0],
        ["BCA",0,0,0,0,0,0],
        ["CAB",0,0,0,0,0,0],
        ["CBA",0,0,0,0,0,0]]
for rating1 in range(1,len(distT)):                    #for
    every row of the array
    for rating2 in range(1,len(distT[0])):     #for
        every cell of the row
        c=0                                            #sum of
            compatibles
        d=0                                            #sum of
            incompatibles
        n=0                                            #pair sum
        if ((distT[rating1][0].find("A")<distT[rating1
            ][0].find("B") and distT[0][rating2].find("A
            ")<distT[0][rating2].find("B")) or (distT[
            rating1][0].find("A")>distT[rating1][0].find
            ("B") and distT[0][rating2].find("A")>distT
            [0][rating2].find("B"))):
                c+=1 #compatible increase
        elif ((distT[rating1][0].find("A")<distT[
            rating1][0].find("B") and distT[0][rating2].
            find("A")>distT[0][rating2].find("B")) or (
            distT[rating1][0].find("A")>distT[rating1
            ][0].find("B") and distT[0][rating2].find("A
            ")<distT[0][rating2].find("B"))):
                d+=1 #or if the two compare and not the
                    same way then they are incompatible
n+=1
        if ((distT[rating1][0].find("A")<distT[rating1
            ][0].find("C") and distT[0][rating2].find("A
            ") < distT[0][rating2].find("C")) or (distT[
            rating1][0].find("A") > distT[rating1][0].
            find("C") and distT[0][rating2].find("A")>
            distT[0][rating2].find("C"))):
                c+=1
        elif ((distT[rating1][0].find("A")<distT[
            rating1][0].find("C") and distT[0][rating2].
            find("A") > distT[0][rating2].find("C")) or
            (distT[rating1][0].find("A") > distT[rating1
```

```
            ][0].find("C") and distT[0][rating2].find("A
            ")<distT[0][rating2].find("C"))):
                d+=1
#same for A and C
        n+=1
        if ((distT[rating1][0].find("B")<distT[rating1
            ][0].find("C") and distT[0][rating2].find("B
            ") < distT[0][rating2].find("C")) or (distT[
            rating1][0].find("B") > distT[rating1][0].
            find("C") and distT[0][rating2].find("B")>
            distT[0][rating2].find("C"))):
                c+=1
        elif ((distT[rating1][0].find("B")<distT[
            rating1][0].find("C") and distT[0][rating2].
            find("B") > distT[0][rating2].find("C")) or
            (distT[rating1][0].find("B") > distT[rating1
            ][0].find("C") and distT[0][rating2].find("B
            ")<distT[0][rating2].find("C"))):
                d+=1
#same for B and C, calculating the tau distance
    rounding it at the second decimal
        n+=1
                #print((distT[rating1][0].find("A") <
                    distT[rating1][0].find("B")) and (
                    distT[0][rating2].find("A") < distT
                    [0][rating2].find("B")) or(distT[
                    rating1][0].find("A") > distT[
                    rating1][0].find("B")) and (distT
                    [0][rating2].find("A") > distT[0][
                    rating2].find("B")))
                #print((distT[rating1][0].find("A") <
                    distT[rating1][0].find("C")) and (
                    distT[0][rating2].find("A") < distT
                    [0][rating2].find("C")) or (distT[
                    rating1][0].find("A") > distT[
                    rating1][0].find("C")) and (distT
                    [0][rating2].find("A") > distT[0][
                    rating2].find("C")),)
                #print((distT[rating1][0].find("B") <
                    distT[rating1][0].find("C")) and (
                    distT[0][rating2].find("B") < distT
```

```
        [0][rating2].find("C")) or (distT[
        rating1][0].find("B") > distT[
        rating1][0].find("C")) and (distT
        [0][rating2].find("B") > distT[0][
        rating2].find("C")))
    distT[rating1][rating2]=round((c-d)/n
        ,2)
```

6.1.2.2 Calculation of 5 and 2 nearest values.

In this subsection the selection of the nearest values is being done. More specifically, first the tables containing the 5 nearest values (near5) ,the pairs corresponding to the 5 nearest values (near5L), the 2 nearest values (near2) and the pairs corresponding to the 2 nearest values (near2L) are initialized. Then, for each cell in the table of pair relations, it checks whether the pair corresponds to one of the pairs to be compared(ABC,BCA,BAC) and if it is greater than the smallest value in the table of 5 or 2 closest, it stores its value and the pairs to which this value corresponds.

Listing 6.7: Calculating the 2 and 5 closest correlations

```
near5 = [["ABC",-10,-9.9,-10.1,-10.2,-9.8],
         ["BAC",-10,-9.9,-10.1,-10.2,-9.8],
         ["BCA",-10,-9.9,-10.1,-10.2,-9.8]]
#the array with the 5 closest values
near5L=[["␣","␣","␣","␣","␣"],
         ["␣","␣","␣","␣","␣"],
         ["␣","␣","␣","␣","␣"]]
#the array containing the elements of the 5 closest
    values
finalR5=[["ABC",-9],
         ["ACB",-9],
         ["BAC",-9],
         ["BCA",-9],
         ["CAB",-9],
         ["CBA",-9]]
near2L= [["␣","␣"],
         ["␣","␣"],
         ["␣","␣"]]
#the element array with the two closest value
near2= [["ABC",-9,-8],
         ["BAC",-9,-8],
         ["BCA",-9,-8]]
#the value array with the two closest values
finalR2=[["ABC",0],
         ["ACB",0],
         ["BAC",0],
         ["BCA",0],
         ["CAB",0],
         ["CBA",0]]
```

```
for group in range(1,len(distT)):
        for dist in range(group+1,len(distT)):
                distT[group][dist]=-1
#initialization of the array's diagonal with -1
#For every row, for every cell, if not in the diagonal
for group in range(1,len(distT)):
    for dist in range(1,len(distT[group])):
        if dist != group:
            #if the element of row or column is the
               first of near5
             if distT[group][0]==near5[0][0] or
                distT[0][dist]==near5[0][0]:
             #if the distance is less than the
                smallest element of array's row
                    if distT[group][dist]>min(near5
                       [0][1:]):
                     #if equal to row's element
                                if distT[group][0]==
                                   near5[0][0]:
                                        near5L[0][near5
                                           [0].index(
                                           min(near5
                                           [0][1:]))
                                           -1]=distT
                                           [0][dist]
                     #if equal to column's element
                                elif distT[0][dist]==
                                   near5[0][0]:
                                        near5L[0][near5
                                           [0].index(
                                           min(near5
                                           [0][1:]))
                                           -1]=distT[
                                           group][0]
                                near5[0][near5[0].index
                                   (min(near5[0][1:]))
                                   ]=distT[group][dist]
                                for g in range(0,len(
                                   finalR5)):
                                        if finalR5[g
                                           ][0]==distT[
```

```
                                                group][0]:
                                                    finalR5
                                                        [g
                                                        ][1]+=
                                                        distT
                                                        [
                                                        group
                                                        ][
                                                        dist
                                                        ]
#same for the 2 closest values
                    if distT[group][0]==near2[0][0] or
                       distT[0][dist]==near2[0][0]:
                            if distT[group][dist]>min(near2
                               [0][1:]):
                                    if distT[group][0]==
                                       near2[0][0]:
                                            near2L[0][near2
                                               [0].index(
                                               min(near2
                                               [0][1:]))
                                               -1]=distT
                                               [0][dist]
                                    elif distT[0][dist]==
                                       near2[0][0]:
                                            near2L[0][near2
                                               [0].index(
                                               min(near2
                                               [0][1:]))
                                               -1]=distT[
                                               group][0]
                                    near2[0][near2[0].index
                                       (min(near2[0][1:]))
                                       ]=distT[group][dist]
#the same for the second element of near5
                    if distT[group][0]==near5[1][0] or
                       distT[0][dist]==near5[1][0]:
                            if distT[group][dist]>min(near5
                               [1][1:]):
                                    if distT[group][0]==
                                       near5[1][0]:
```

```
                                near5L[1][near5
                                    [1].index(
                                    min(near5
                                    [1][1:]))
                                    -1]=distT
                                    [0][dist]
                            elif distT[0][dist]==
                                near5[1][0]:
                                    near5L[1][near5
                                    [1].index(
                                    min(near5
                                    [1][1:]))
                                    -1]=distT[
                                    group][0]
                            near5[1][near5[1].index
                                (min(near5[1][1:]))
                                ]=distT[group][dist]
                            for g in range(0,len(
                                finalR5)):
                                    if finalR5[g
                                    ][0]==distT[
                                    group][0]:
                                        finalR5
                                        [g
                                        ][1]+=
                                        distT
                                        [
                                        group
                                        ][
                                        dist
                                        ]

        if distT[group][0]==near2[1][0] or
            distT[0][dist]==near2[1][0]:
                if distT[group][dist]>min(near2
                    [1][1:]):
                        if distT[group][0]==
                            near2[1][0]:
                                near2L[1][near2
                                    [1].index(
                                    min(near2
```

```
                                                    [1][1:]))
                                                    -1]=distT
                                                    [0][dist]
                                            elif distT[0][dist]==
                                                near2[1][0]:
                                                    near2L[1][near2
                                                    [1].index(
                                                    min(near2
                                                    [1][1:]))
                                                    -1]=distT[
                                                    group][0]
                                            near2[1][near2[1].index
                                                (min(near2[1][1:]))
                                                ]=distT[group][dist]
#same thing for the third element
                                    if distT[group][0]==near5[2][0] or
                                        distT[0][dist]==near5[2][0]:
                                            if distT[group][dist]>min(near5
                                                [2][1:]):
                                                    if distT[group][0]==
                                                        near5[2][0]:
                                                            near5L[2][near5
                                                            [2].index(
                                                            min(near5
                                                            [2][1:]))
                                                            -1]=distT
                                                            [0][dist]
                                                    elif distT[0][dist]==
                                                        near5[2][0]:
                                                            near5L[2][near5
                                                            [2].index(
                                                            min(near5
                                                            [2][1:]))
                                                            -1]=distT[
                                                            group][0]
                                                    near5[2][near5[2].index
                                                        (min(near5[2][1:]))
                                                        ]=distT[group][dist]
                                                    for g in range(0,len(
                                                        finalR5)):
```

```
                                    if finalR5[g
                                       ][0]==distT[
                                       group][0]:
                                            finalR5
                                               [g
                                               ][1]+=
                                               distT
                                               [
                                               group
                                               ][
                                               dist
                                               ]
            if distT[group][0]==near2[2][0] or
               distT[0][dist]==near2[2][0]:
                     if distT[group][dist]>min(near2
                        [2][1:]):
                           if distT[group][0]==
                              near2[2][0]:
                                 near2L[2][near2
                                    [2].index(
                                    min(near2
                                    [2][1:]))
                                    -1]=distT
                                    [0][dist]
                           elif distT[0][dist]==
                              near2[2][0]:
                                 near2L[2][near2
                                    [2].index(
                                    min(near2
                                    [2][1:]))
                                    -1]=distT[
                                    group][0]
                           near2[2][near2[2].index
                              (min(near2[2][1:]))
                              ]=distT[group][dist]
```

6.1.2.3 Calculation of Kemeny-optimal value

In the last step the Kemeny-optimal value is calculated. Specifically, for each unique item in the list with the 2 closest values, it calculates the number that appears in the lists of the 3 users. The same procedure is done for the 5 closest ones with the difference of calculating the artifact of each element.

Listing 6.8: Calculation of Kemeny-optimal value

```
#list printing
for a in distT:
        print(a)
        print()
print("---Near 2 list ---")
for i in near2:
        print(i)
        print()
for i in near2L:
        print(i)
        print()

print("---Near 5 list ---")
for i in near5:
        print(i)
        print()
for i in near5L:
        print(i)
        print()
u2=[]
u5=[]
#selecting the unique list's elements for the nearest 2
    and 5
for g in near2L:
        for i in g:
                if i not in u2:
                        u2.append(i)
for s in near5L:
        for i in s:
                if i not in u5:
                        u5.append(i)
#count the elements
for i in range(0,len(u2)):
```

```
    cnt=0
    for k in range(0,len(near2L)):
        for j in range(0,len(near2L[k])):
                if near2L[k][j]==u2[i]:
                        cnt+=1
    print(f"{u2[i]} : {cnt}")
print()
#summary of every element
for i in range(0,len(u5)):
    cnt=0
    for k in range(0,len(near5L)):
        for j in range(0,len(near5L[k])):
            if near5L[k][j]==u5[i]:
                cnt+=near5[k][j+1]
    print(f"{u5[i]} : {cnt}")
```

6.1.2.4 Calculation of the central value of ratings

In this step the calculation of the centroid values of the ratings is done. After initializing the table, the average values of each user are calculated and then the average value of each user's rating is subtracted.

Listing 6.9: Array initialization and calculating users' mean value

```
ratings = [[0,2,0,1,2,2,0,0,0,1],       #Giannis
           [1,2,2,0,0,2,1,1,0,0],       #Mary
           [1,1,1,0,0,2,1,0,1,1],       #Kwstas
           [0,0,2,0,0,1,0,0,0,0],       #Nikoleta
           [2,0,0,2,0,2,2,2,1,0],       #Georgia
           [0,2,1,1,0,2,0,2,1,1]]       #Katerina
m_o=[]
for user in ratings:                                #for
    every user
    count=0
    sum=0
    for rating in user:                           #for
       every rating
        if rating!=0:
            count+=1
            sum+=rating                     #summary
                calculation
```

```
    m_o.append(round(sum/count,2))                    #mean
        value
print(m_o)
```

6.1.2.5 Calculation of the central value for each evaluation

The next step is to subtract the average value of each user from their reviews.

Listing 6.10: Calculating the centroid values for each user

```
mean_centered=ratings
for user in ratings:
    for rating in user:
        if rating!=0:
            mean_centered[ratings.index(user)][user.
                index(rating)]=round(rating-m_o[ratings.
                index(user)],2)
for user in ratings:
    print(user)
    print()
```

6.1.2.6 Calculation of weights for the propagation path

In this step, the weights are calculated at the user and book-article level

Listing 6.11: Calculating users' weight

```
import math
import itertools
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

#Class - users
class person:
    def __init__(self, name, job, attributes, node):
        self.name = name
        self.job = job
        self.attributes = attributes
        self.node=node

#Class - items
class book:
```

```
    def __init__(self, name, fields, node):
        self.name = name
        self.fields = fields
        self.node = node

#Function - returns true if attr is in list
def isIn( attr , list):
    found=False
    for elem in list:
        if elem[0]==attr:
            return True
    return False

def sigmoid(array):
    return 1 / (1 + math.exp(-array) )

#Function to calculate the weights
def calcWeights ( ):
    print()
    print(l_coop)
    print()
    for cooperation in l_coop:
        cnt_interests=0
        for interest in cooperation[0].interests:
            for interest2 in cooperation[1].interests:
                if interest==interest2:
                    print('Found a common interest:',
                        interest,'in users :',cooperation
                        [0].name, ', ',cooperation[1].name)
                    cnt_interests+=1
                    for edge in G.edges.data():
                        if edge[0]==cooperation[0].node and
                            edge[1]==cooperation[1].node:
                            G.add_edge(cooperation[0].node,
                                cooperation[1].node, weight=
                                cnt_interests)
                            print('node:',cooperation[0].node,
                                ', ',cooperation[1].node)
        if cnt_interests==0:
            G.add_edge(cooperation[0].node, cooperation
                [1].node, weight=0)
```

```
            print('No common interests for ',cooperation
                [0].name, ' - ',cooperation[1].name, ',
                assign 0 weight')
    for like in l_likes:
        print ('Like', like[0].name ,' - ', like[1].
            fields)
    for item in l_item:
        cnt=0
        for like in l_likes:
            if like[1]==item:
                cnt+=1
                print('like :',like[0].node, ' ' , like[1].
                    node)
                for user in l_users:
                    if user.name == like[0].node:
                        w_user_items[l_item.index(item)][
                            l_users.index(user)] = 1
        for edge in G.edges.data():
            if edge[1]==item.node:
                print(edge)
                G.add_edge(edge[0],edge[1],weight=cnt)

#Function to calculate the vector of calcPerson
def calcInterests ( calcPerson ):
    print()
    print('START: Calculating the vector for user: ',
        calcPerson.name)
    print()

    neighbors = [n for n in G.neighbors(calcPerson.name)
        if G.edges[calcPerson.name, n]['type'] == 'likes
        ']
    print('Peter s neigbhors')
    print(neighbors)
    print(' ')
    print()
    print('printing v_attributes')
    print(v_attributes)

    h_i = G.nodes[calcPerson.name]['attributes']  #
        Initial features of the node
```

```
    vector_features = []

    for neighbor in neighbors:
        print()
        print('printing neighbor')
        print(neighbor)
        print()
        print('printing user_attributes')
        print(v_user_attributes)
        print()
        q_j=[]
        for match in v_match:
            if match[0] == neighbor:
                q_j = v_all_attributes[match[1]]
                print('printing q_j')
                print(q_j)
                print()
        vector_features = vector_features + q_j
    print()
    print('printing the vector_featurs')
    print(vector_features)
    product = np.dot(w_user_items, vector_features)
    print()
    print('printing user weights')
    print(w_user_items)
    print()
    print('printing product')
    print(product)
    print()
    for i in range(0,len(product)):
        tmp_prod = sigmoid(product[i])
        sigmoid_product.append(tmp_prod)

    print ('')
    print('**vector features**')
    print(vector_features)
    print('')
    #print('**final product**')

    return(sigmoid_product)
```

```
l_likes = []                # All the book likes of users
l_item = []                 # All the books
l_users = []                # All the users
l_coop = []                 # All the user cooperations
w_items=[]                  #weight of Books to wanted
    user
w_users=[]                  #weigth of users to wanted
    user
w_user_items=[ [0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0,0] ,
    [0,0,0,0,0,0,0,0,0,0], [0,0,0,0,0,0,0,0,0,0]]
w_user_edges=[]
common_books=[]

v_attributes = []
v_features = []
v_user_attributes = []
v_all_attributes = []
vector_features = []
v_match = []
sigmoid_product = []
#node 2

##
## NODES AND EDGES
G = nx.DiGraph( )
G.add_node('Peter',t='person')
G.add_nodes_from(['Peter'], name='Peter')
G.add_nodes_from(['Peter'],prof='')
G.add_nodes_from(['Peter'],attributes=[])
G.add_node('Nick',t='person')
G.add_nodes_from(['Nick'], name='Nick')
G.add_nodes_from(['Nick'],prof='Professor')
G.add_nodes_from(['Nick'],attributes=['Machine␣Learning
    '])
G.add_node('Mary',t='person')
G.add_nodes_from(['Mary'], name='Mary')
G.add_nodes_from(['Mary'],prof='Professor')
G.add_nodes_from(['Mary'],attributes=['Machine␣Learning
    ','Neural␣Networks'])
G.add_node('Article1',t='book')
```

```
G.add_nodes_from(['Article1'], attributes=['Artificial␣
    Intelligence','Machine␣Learning'])
G.add_node('Article2',t='book')
G.add_nodes_from(['Article2'], attributes=['Artificial␣
    Intelligence','Machine␣Learning','Neural␣Networks'])
G.add_node('Article3',t='book')
G.add_nodes_from(['Article3'], attributes=['Nature','
    Biology'])
edges_likes = [('Peter','Article1'),('Peter','Article3'
    ),('Nick','Article1'),('Mary','Article2')]
edges_collaboration = [('Mary','Nick'),('Nick','Peter')
    ]
G.add_edges_from(edges_likes,type='likes')
G.add_edges_from(edges_collaboration,type='
    collaboration')
node_colors = ['lightblue' if G.nodes[node]['t'] == '
    person' else 'orange' for node in G.nodes()]
edge_colors = ['red' if G[u][v]['type'] == 'likes' else
     'green' for u, v in G.edges()]
positions = {'Peter' :[2 ,5] , 'Nick' :[4 ,5] ,'Mary'
    :[6 ,5] , 'Article1' :[4 ,1] , 'Article2' :[6 ,1] ,'
    Article3' :[2 ,1]}
nx . draw_networkx (G, node_size =600 , with_labels =
    True , node_color=node_colors , edge_color=
    edge_colors, pos=positions )
plt.show()


node_color=node_colors,
## END NODES AND EDGES
##

##
## END INPUTS BUT IN GRAPHS

##
## CREATION OF LISTS
#Users with their interests
for node in G.nodes.data():
    if node[1]['t']=='person':
```

```
            l_users.append(person(node[1]['name'],node[1]['
                prof'],node[1]['attributes'],node[0] ) )
        else:
            l_item.append(book(node[0],node[1]['attributes'],
                node[0] ) )
        for attribute in node[1]['attributes']:
            if attribute not in v_attributes:
                v_attributes.append(attribute)

for user in l_users:
    #print()
    v_user_attributes.append([])
    v_all_attributes.append([])
    v_match.append([user.name,l_users.index(user)])
    for item in v_attributes:
        v_user_attributes[l_users.index(user)].append(0)
        v_all_attributes[l_users.index(user)].append(0)
    for attribute in v_attributes:
        if attribute in user.attributes:
            print('user attribute found')
            v_user_attributes[l_users.index(user)][
                v_attributes.index(attribute)] = 1;
            v_all_attributes[l_users.index(user)][
                v_attributes.index(attribute)] = 1;
for item in l_item:
    v_all_attributes.append([])
    v_match.append([item.name,l_item.index(item)+len(
        l_users)])
    for attribute in v_attributes:
        v_all_attributes[l_item.index(item)+len(l_users)
            ].append(0)
    for attribute in v_attributes:
        if attribute in item.fields:
            print('item attribute found')
            v_all_attributes[l_item.index(item)+len(
                l_users)][v_attributes.index(attribute)] =
                1;
## END CREATION OF LISTS

#For to append to list of cooperations between users
    from G.node and likes of users to lists
```

```
for edge in G.edges:
    if (G.nodes.data()[edge[0]])['t']=='person' and (G.
       nodes.data()[edge[1]])['t']=='person' :
        print('Found a user-user relationship',(G.nodes.
           data()[edge[0]])['name'],' - ',(G.nodes.data
           ()[edge[1]])['name'])
    if (G.nodes.data()[edge[0]])['t']=='person' and (G.
       nodes.data()[edge[1]])['t']!='person' :
        for item in l_item:
            if item.node==edge[1]:
                for user in l_users:
                    if user.name == edge[0]:
                        l_likes.append([user,item])

calcWeights()
peterInterests = calcInterests ( l_users[0] )
print()
print('printing the final peter interests vector')
print(peterInterests)

threshold = 0.51
interested_in = [topic for idx, topic in enumerate(
    v_attributes) if peterInterests[idx] >= threshold]
print("Peter is interested in:", " and ".join(
    interested_in))
```

Listing 6.12: Calculating popularity bias in group recommendation system

```
import math
import sys
import itertools
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import math
import re
import random
import copy

#Class - users
```

```
class person:
    def __init__(self, id, age, gender, occupation,
        zip_code):
          self.id = id
          self.age = age
          self.gender = gender
          self.occupation=occupation
          self.zip_code=zip_code
          self.numRatings = 0
          self.numPopularRatings = 0
          self.group = None
          self.popularRatio = 0
          self.ratings = []
          self.numItems = 0

class item:
    def __init__(self, id, title, releaseDate,
        videoReleaseDate, imdbUrl, genres):
          self.id = id
          self.title = title
          self.releaseDate = releaseDate
          self.videoReleaseDate=videoReleaseDate
          self.imdbUrl = imdbUrl
          self.genres = genres
          self.numRatings = 0
          self.numRatingsPredictions = 0
          self.popularity = 0
          self.predictionPopularity = 0

user_group = 100
predictions = 4
path ="movielens100k"
l_groups = ['niche','blockbusters','diverse']
l_groupNum = [0]*len(l_groups)

numUsers = 0
numItems = 0
numRatings = 0

v_genres = []
v_occupations = []
```

```
l_users = []
l_items = []

l_ratings = []

l_usersPopularRatio = [ ]
l_user_items = []

print('Starting calculating with user group: ',
   user_group)
print('Reading the info file ...')
f_info = open(path+r"/u.info", "r")
for line in f_info:
  splitLine = line.split()
  match splitLine[1]:
    case 'users':
      numUsers = int(splitLine[0])
      print('Found: ',numUsers, ' users')
    case 'items':
      numItems = int(splitLine[0])
      print('Found: ',numItems, ' items')
    case 'ratings':
      numRatings = int(splitLine[0])
      print('Found: ',numRatings, ' ratings')
f_info.close()
print('Info file read successfully')

l_user_items = [0]*numUsers
l_user_items_prediction = [0]*numUsers
l_user_items_mitigated_prediction = [0]*numUsers
for user in l_user_items:
  l_user_items[l_user_items.index(user)] = [0]*numItems
for user in l_user_items_prediction:
  l_user_items_prediction[l_user_items_prediction.index
     (user)] = [0]*numItems

print('Reading the genres file ...')
f_genres =  open(path+r"/u.genre", "r")
for line in f_genres:
  splitLine = line.split("|")
```

```
    v_genres.append([splitLine[0],splitLine[1]])
f_genres.close()
print('Genres file read successfully')

print('Reading the occupation file...')
f_occupation = open(path+r"/u.occupation", "r")
for line in f_occupation:
    v_occupations.append(line)
f_occupation.close()
print('Occupation file read successfully')

print('Reading the users file...')
f_users = open(path+r"/u.user", "r")
for line in f_users:
    splitLine = line.split("|")
    l_users.append(person(splitLine[0],splitLine[1],
        splitLine[2],splitLine[3],splitLine[4]))
f_users.close()
print('Users file read successfully')

print('Reading the items file...')
f_items = open(path+r"/u.item", "r")
for line in f_items:
    splitLine = line.split("|")
    l_items.append(item(splitLine[0],splitLine[1][:-6],
        splitLine[1][-5:-1],splitLine[2],splitLine[4],
        splitLine[5:23]))
f_items.close()
print('Items file read successfully')

print('Reading the events file...')
f_data = open(path+r"/u.data", "r")
for line in f_data:
    splitLine = line.split()
    l_ratings.append([splitLine[0],splitLine[1],splitLine
        [2],splitLine[3]])
f_data.close()
print('Events file read successfully')

#Creating user and item lists
for rating in l_ratings:
```

```
  for item in l_items:
    if ( rating[1] == item.id ):
      l_items[l_items.index(item)].numRatings+=1
  for user in l_users:
    if ( rating[0] == user.id ):
      l_users[l_users.index(user)].numRatings+=1
      l_users[l_users.index(user)].ratings.append(
         rating)

l_users.sort(key=lambda x: x.numRatings, reverse=True)
l_items.sort(key=lambda x: x.numRatings, reverse=True)

l_userItems = []

for user in l_users:
  for rating in user.ratings:
    for item in l_items:
      if item.id == rating[1]:
        if ( l_items.index(item) < len(l_items)*2/10 ):
          user.numPopularRatings += 1
        l_user_items[l_users.index(user)][l_items.index
           (item)]= float(rating[2])
    l_userItems.append(rating[1])
  user.popularRatio = user.numPopularRatings / user.
     numRatings * 100
  user.numItems = len(set(l_userItems))

l_usersPopularRatio = sorted(l_users, key=lambda x: x.
   popularRatio, reverse=True)

print('Starting categorizing users')
for user in l_usersPopularRatio:
  if l_usersPopularRatio.index(user) < len(
     l_usersPopularRatio) / 5 :
    l_users[l_users.index(user)].group = l_groups.index
       ('blockbusters')
    l_groupNum[l_groups.index('blockbusters')] += 1
  elif l_usersPopularRatio.index(user) > len(
     l_usersPopularRatio)*4 / 5 :
    l_users[l_users.index(user)].group = l_groups.index
       ('niche')
```

```
    l_groupNum[l_groups.index('niche')] += 1
  else :
    l_users[l_users.index(user)].group = l_groups.index
       ('diverse')
    l_groupNum[l_groups.index('diverse')] += 1
print('Users successfully categorized')

for groupNum in l_groupNum:
  print('Number of users of ',l_groups[l_groupNum.index
     (groupNum)],': ',groupNum)

l_user_similarities = [0]*numUsers
for user in l_user_similarities:
  l_user_similarities[l_user_similarities.index(user)]
     = [-99]*numUsers

userMO = [None]*numUsers
#Calculating the users average score
print('Starting calculating users mean value')
for user in l_users:
  sumRatings = 0
  numRatings = 0
  for rating in l_ratings:
    if rating[0] == user.id:
      sumRatings+=int(rating[2])
      numRatings+=1
  userMO[l_users.index(user)] = sumRatings / numRatings
print('Mean value calculated')

print('Starting calculating users similarity')
for usr1 in l_users:
  for usr2 in l_users[l_users.index(usr1):]:
    var1,var2,var3 = 0,0,0
    for item in l_items:
      if   l_user_items[l_users.index(usr1)][l_items.
         index(item)] != 0 and l_users.index(usr1)!=
         l_users.index(usr2):
        var1 += (l_user_items[l_users.index(usr1)][
           l_items.index(item)] - userMO[l_users.index(
           usr1)]) * (l_user_items[l_users.index(usr2)
           ][l_items.index(item)] - userMO[l_users.
```

```
            index(usr2)])
        var2 += (l_user_items[l_users.index(usr1)][
            l_items.index(item)] - userMO[l_users.index(
            usr1)])**2
        var3 += (l_user_items[l_users.index(usr2)][
            l_items.index(item)] - userMO[l_users.index(
            usr2)])**2
    if var2 !=0 and var3 !=0 :
      l_user_similarities[l_users.index(usr1)][l_users.
          index(usr2)] = round( var1/(math.sqrt(var2)*
          math.sqrt(var3)) ,2)
      l_user_similarities[l_users.index(usr2)][l_users.
          index(usr1)] = round( var1/(math.sqrt(var2)*
          math.sqrt(var3)) ,2)
print('User similarity calculated successfully')


print('Finding 3 most similar users for every user')
top_similarities_index=[None]*numUsers
top_similarities_value=[0]*numUsers
for item in top_similarities_index:
  top_similarities_index[top_similarities_index.index(
      item)] = [None]*user_group
for item in top_similarities_value:
  top_similarities_value[top_similarities_value.index(
      item)] = [-99]*user_group
for user_similarities in l_user_similarities:
  sim_idx = 0
  for sim in user_similarities:
    if sim>min(top_similarities_value[
        l_user_similarities.index(user_similarities)]):
      tmp_idx = top_similarities_value[
          l_user_similarities.index(user_similarities)].
          index(min(top_similarities_value[
          l_user_similarities.index(user_similarities)])
          )
      top_similarities_value[l_user_similarities.index(
          user_similarities)][tmp_idx] = sim
      top_similarities_index[l_user_similarities.index(
          user_similarities)][tmp_idx] = sim_idx
    sim_idx+=1
```

```
#Finding the prediction values
l_user_items_prediction = copy.deepcopy(l_user_items)

for user_ratings in l_user_items_prediction:
  idx=0
  for rating in user_ratings:
    if rating==0:
      var1=0
      var2=0
      for top_sim in top_similarities_index[
         l_user_items_prediction.index(user_ratings)]:
        if top_similarities_value[
           l_user_items_prediction.index(user_ratings)
           ][top_similarities_index[
           l_user_items_prediction.index(user_ratings)
           ].index(top_sim)]!= 0 :
          var1+=top_similarities_value[
             l_user_items_prediction.index(user_ratings
             )][top_similarities_index[
             l_user_items_prediction.index(user_ratings
             )].index(top_sim)] * ( l_user_items[
             top_sim][idx] - userMO[top_sim] )
          var2+=top_similarities_value[
             l_user_items_prediction.index(user_ratings
             )][top_similarities_index[
             l_user_items_prediction.index(user_ratings
             )].index(top_sim)]
      if var1!=0 and var2!=0:
        l_user_items_prediction[l_user_items_prediction
           .index(user_ratings)][idx]=round(userMO[
           l_user_items_prediction.index(user_ratings)
           ]+var1/var2 , 2 )
    idx+=1
print('User prediction ratings calculated')

#making a list of the top ratings
l_max_predictions=[0]*numUsers
l_max_predictions_index=[0]*numUsers
l_max_predictions_mitigated=[0]*numUsers
l_max_predictions_mitigated_index=[0]*numUsers
```

```
for user_ratings in l_user_items:
  l_max_predictions[l_user_items.index(user_ratings)] =
      [-99]*predictions
  l_max_predictions_index[l_user_items.index(
     user_ratings)] = [None]*predictions
  l_max_predictions_mitigated[l_user_items.index(
     user_ratings)] = [-99]*predictions
  l_max_predictions_mitigated_index[l_user_items.index(
     user_ratings)] = [None]*predictions
  idx = 0
  for rating in user_ratings:
    if l_user_items_prediction[l_user_items.index(
       user_ratings)][idx] != 0 and l_user_items[
       l_user_items.index(user_ratings)][user_ratings.
       index(rating)]==0 and l_user_items_prediction[
       l_user_items.index(user_ratings)][idx] > min(
       l_max_predictions[l_user_items.index(
       user_ratings)]):
      idx_pred = l_max_predictions[l_user_items.index(
         user_ratings)].index(min(l_max_predictions[
         l_user_items.index(user_ratings)]))
      l_max_predictions[l_user_items.index(user_ratings
         )][idx_pred] = l_user_items_prediction[
         l_user_items.index(user_ratings)][idx]
      l_max_predictions_index[l_user_items.index(
         user_ratings)][idx_pred] = idx
    idx+=1
l_max_predictions_mitigated = copy.deepcopy(
   l_max_predictions)
l_max_predictions_mitigated_index = copy.deepcopy(
   l_max_predictions_index)
for item in l_max_predictions_mitigated_index:
  for replaced_prediction in item[-int(len(item)/3):]:
    l_max_predictions_mitigated_index[
       l_max_predictions_mitigated_index.index(item)][
       item.index(replaced_prediction)] = random.
       randrange(len(l_items))

average_movie_popularity = [-10]*len(l_groups)
for item in l_items:
  item.popularity = item.numRatings/numUsers
```

```
#CF-user collaborative
var1_groups = 0
var2_groups = 0
var1r_groups = 0
var2r_groups = 0
var1_final = 0
var1r_final = 0
GAP_groups = [0]*len(l_groups)
GAPr_groups = [0]*len(l_groups)
#MPIR
popular_items = sorted(l_items, key=lambda x: x.
    popularity, reverse=True)
var1b_groups = 0
var2b_groups = 0
var1b_final = 0
#Random
rand_recommendations = [0,0,0,0]*len(l_users)
var1c_groups = 0
var2c_groups = 0
var1c_final = 0
var1d_groups = 0
var2d_groups = 0
var1d_final = 0
GAPbr_groups = [0]*len(l_groups)
GAPcr_groups = [0]*len(l_groups)
GAPdr_groups = [0]*len(l_groups)

for rec in rand_recommendations:
  rand_recommendations[rand_recommendations.index(rec)
      ]=[random.randrange(len(l_items)),random.randrange
      (len(l_items)),random.randrange(len(l_items)),
      random.randrange(len(l_items))]

print('Starting calculating GAP values')
#FOR every group
for group in l_groups:
  g=0
  var2_groups=0
  var2r_groups=0
  var2b_groups=0
```

```
var2c_groups=0
#FOR every user
for user_ratings in l_user_items_prediction:
  b=0
  if l_users[l_user_items_prediction.index(
     user_ratings)].group == l_groups.index(group):
    p1=0
    p2=0
    p3=0
    p4=0
    var1_groups=0
    var1r_groups=0
    var1b_groups=0
    var1c_groups=0
    #FOR every rating
    for rating in range(len(user_ratings)):
      if l_user_items[l_user_items_prediction.index(
         user_ratings)][rating]!=0 :
        var1_groups += l_items[rating].popularity
        p1+=1
      if l_user_items[l_user_items_prediction.index(
         user_ratings)][rating]==0 and user_ratings[
         rating] != 0 and ( rating in
         l_max_predictions_index[
         l_user_items_prediction.index(user_ratings)]
          ) :
        var1r_groups += l_items[rating].popularity
        p2+=1
      if l_items[rating] in popular_items[:3]:
        var1b_groups+= l_items[rating].popularity
        p3+=1
      b+=1
      for rec in rand_recommendations[
         l_user_items_prediction.index(user_ratings)
         ]:
        if rating == rec:
          var1c_groups+= l_items[rating].popularity
          p4+=1
    if p1!=0:
      var2_groups+=var1_groups/p1
    if p2!=0:
```

```
            var2r_groups+=var1r_groups/p2
          if p3!=0:
            var2b_groups+=var1b_groups/p3
          if p4!=0:
            var2c_groups+=var1c_groups/p4
          g+=1
    GAP_groups[l_groups.index(group)] = var2_groups/g
    GAPr_groups[l_groups.index(group)] = var2r_groups/g
    GAPbr_groups[l_groups.index(group)] = var2b_groups/g
    GAPcr_groups[l_groups.index(group)] = var2c_groups/g
print('Finished calculating GAP values')

DGAP_values = [0]*len(GAP_groups)
DGAPb_values = [0]*len(GAP_groups)
DGAPc_values = [0]*len(GAP_groups)
print('GAP values of the 3 groups')
print (GAP_groups)
print (GAPr_groups)
print (GAPbr_groups)
print (GAPcr_groups)
print('GAP of recommendation of the 3 groups')
for group in l_groups:
    DGAP_values[l_groups.index(group)] = (GAPr_groups[
        l_groups.index(group)]-GAP_groups[l_groups.index(
        group)])/GAP_groups[l_groups.index(group)]
    print('DGAP of User-based CF group:', l_groups[
        l_groups.index(group)],' ',DGAP_values[l_groups.
        index(group)])
for group in l_groups:
    DGAPb_values[l_groups.index(group)] = (GAPbr_groups[
        l_groups.index(group)]-GAP_groups[l_groups.index(
        group)])/GAP_groups[l_groups.index(group)]
    print('DGAP of MPIR group:', l_groups[l_groups.index(
        group)],' ',DGAPb_values[l_groups.index(group)])
for group in l_groups:
    DGAPc_values[l_groups.index(group)] = (GAPcr_groups[
        l_groups.index(group)]-GAP_groups[l_groups.index(
        group)])/GAP_groups[l_groups.index(group)]
    print('DGAP of RANDOM  group:', l_groups[l_groups.
        index(group)],' ',DGAPc_values[l_groups.index(
        group)])
```