THE FACULTIES OF ENGINEERING, SCIENCE AND MEDICINE - AALBORG UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

AALBORG UNIVERSITY

DE4 – GROUP D624A

Collaborative Filtering in Social Networks

Similarity analysis and feedback techniques

Sergio Mateo María

27/05/2010

The Faculties of Engineering, Science and Medicine

Aalborg University

Department of Computer Science

TITLE:

Collaborative Filtering in Social Networks. Similarity analysis and feedback techniques

PROJECT PERIOD: DE4,
February 2nd 2010 - May 31th 2010

PROJECT GROUP: D624A

AUTHOR: Sergio Mateo

SUPERVISOR: Ira Assent

NUMBER OF COPIES: 3

REPORT PAGES: 84

ABSTRACT

Social networks become an important way to provide information to recommender systems that use collaborative filtering.

In this work, we propose and analyze the results of many methods to calculate the similarity between users, in different scenarios.

In other way, we describe some techniques that allow the users to give feedback to the system in order to have influence in the recommendations they receive.

Signature:

Table of Contents

Index of formulas

Index of examples

Chapter 1. Introduction.

Nowadays, we are witnessing in the expansion of the information on the Internet. All the information we need about a specific topic is available in the network, but in many cases the problem is the difficulty to find the information useful for us, among big amounts of useless one. The origin of this problem is that the information is unstructured. As a result it can be very hard to find relevant elements of a specific category of item, such as movies, music, thematic web pages, ...

One possibility is to use recommender systems that clarify the search of relevant items between all the available information, using the opinion of other people to know which of the items are really relevant [AT05].

The research in this scope has discovered many methods to get, through the opinion of other people, the relevant items for a specific person. The most of these methods work around the idea of finding similarities in the taste of the people, using historical information of them about preferences. Then, the prediction for a specific person is based in the opinion of the most similar user to the person. This procedure is known as *Collaborative Filtering*.

It is important that these systems can improve their results in the recommendation task with the *feedback* their users can give. For this reason, it is necessary the users understand how the recommendations network works. Then, the users can make changes over the recommendations they receive.

In the last years, the importance of the social networks is growing and they become relevant for research studies. The information provided by users in social networks is a powerful source of information for the recommender systems in the user's profiles and also in the links that exist in the structure of the network [A09].

The idea of this project is to analyze different measures of similarity between users and which are used in the recommendations in the collaborative filtering systems and to study some techniques which get feedback from the users of a recommender system.

PART I

ANALYSIS

Chapter 2. Application Context.

Some applications are designed under a set of requirements which create the necessity to develop the system in a specific way.

In this case, the developed application does not follow the requirements of a customer. The target of this application is to aid the study of different procedures in the scope of the recommender system that uses collaborative filtering. These procedures can be integrated in the future in social networks and make recommendations to the users that match their personal taste.

The developed application loads data from users, movies and the values of ratings that the users give to some of the movies. With this information, the application calculates the similarities between users with different procedures, recommends films to the users and can modify the recommendations using the feedback the user gives.

All the processed data is saved in files in order to analyze later which of the similarity calculation procedures works better or how the feedback of the user changes the recommendations.

Chapter 3. Social Networks Sites.

The social networks sites (SNS) are web-based services where the user can define a profile [BE97]. These sites let users start a relationship between strangers, but mostly the SNS are for the users a way to show other people their offline social network. In this case, the main target of the users is to communicate with others SNS users, with whom they have a connection [BE97].

The users can decide the privacy of each one of the items of this profile. To manage the privacy, each user sets a connection with other SNS users, and the items of his or her profile the user wants to share.

When a user has joined a SNS, he or she can find other people in the system, to have an online relationship. The tag used for these relationships is different for each SNS: “Friends”, “Contacts”, “Follow/Followers”, “Fans”.

The public display of these relationships is important for the users, because it lets the SNS user navigate through the friend’s profile by clicking in the friends list.

The user can navigate through his or her contact list and view the profile of contact. Most of the SNS allow users to send messages to the profiles of the contact list in two ways [BE97]:

- Private message that only the recipient can see.
- Public message that appear in the recipient’s profile and all his or her contacts can read.

One of the most popular things in the major SNS nowadays is the multimedia content. In particular, the image and video sharing [BE97].

Another utility provided by SNS is the possibility to manage events. Users can create a profile for an event, with a description, a place, a day and a time. The people can invite the contacts to the event, and the people invited can decide if they accept, deny or maybe assist.

Users are interested in growing their own networks of relationships [A09]. It is common that one user has hundreds of online relationships, some of them that come from offline relations, maybe some of them come from online relations because they share common interests. It is possible to let user organize the friends/contacts in groups. With groups is possible to send a message to the entire group, share multimedia content with them.

One of the most important aspects of the SNS is the care taken in the privacy policy. Some users are reluctant to share private information online [A09].

Most of the SNS provide many options to manage the privacy of each item (photo, comment, video, …) shared by the user in his or her profile. Usually, users can choose to share the information with all the people in the SNS, only with his or her contacts or with specific groups of contacts.

Chapter 4. Recommender Systems.

Recommender Systems are a technology that automates the process of suggesting items (such as films, music, news, pictures, web pages, etc) that can be interesting for a specific user of the system. For this prediction, the system uses historical information from this user and the other users of the system. If the system doesn't have historical information about the user, it can try to get an initial prediction with the user preferences as source.

The system has to be independent from the content it is recommending. This means that it is not necessary that the system knows which kind of items it is recommending. The same system should be able to recommend music, films or books if it has past ratings of these kinds of items.

There is a common functionality for the recommender systems. The basic tasks that these systems have to offer to users are [SFHS97]:

- First of all the system has to recommend a list of items, that the system considers the most useful for the specific user.
- In other cases when a user asks for an item, the system has to calculate the predicted rating of the item for this specific user.
- And finally if the user sets many restrictions that generate a particular set of items, the system has to be able to recommend from within that set.

To get this functionality, most of the recommender systems work in three steps [BM05]:

- The user provides to the system information about his or her preferences, likes/dislikes. It can be in two ways.
  - Explicit information if, for example, the user gives a rating for a specific item.
  - Implicit information, if the system saves how many times a user listens a song.
- All the information collected in the first step is used to create a profile of user preferences.
- The system computes the profiles of the users to get recommendations for them.

There are three paradigms for collecting and computing the information:

- Content-based: The system recommends to the user, using historical information of the user preferences and using an evaluation of the content of the items to recommend.
- Collaborative filtering: The system recommends to the user, using the preferences of user with similar preferences.
- The third paradigm that consists in a hybrid method of them.

The task of obtaining the information about the preferences of the users is usually done using data mining techniques. Data mining is a technique that consists of extracting implicit information from the data using data analysis and discovery algorithms [EKSWX 98].

In the context of recommended systems in social networks, the purpose is to get information from the profiles of the users with data mining techniques to have information about their preferences and to recommend items based in this information.

4.1. Content-based systems.

The user will receive information similar to that which has shown interest before. The system creates a profile of user preferences based in historical information and recommends to him/her items similar to the items that the user likes in the past.

The content-base systems have to use techniques to qualify the content of the items. It depends of the kind of content we want that the system recommends. Nowadays, many of the content-based systems works with text content (documents and web pages) because they are easier to analyze and qualify the content [AT05].

For example, if the system recommends web sites, it can use the most representative words of the web pages to evaluate the content [BS97].

If the system works with films, maybe it is possible to use information from some cinema organization to classify films. Or it can use objective criteria as the percentage of minutes of the film that there are of each kind of situation (adventure, comedy, drama, crime horror, love). The problem in this case is the qualify task automation of classifying the content.

When the system has the profile of the user and the qualification of the content of the items, compute the utility that the user can get from the items. The items that create the greatest utility for a specific user, they will be the items recommended to him or her.

There are many ways to get a profile of the user:

- For example, with data mining techniques it is possible to analyze the previous ratings given by the users and estimate the importance that they give to each attribute.
- Another possibility is to ask the users for this information directly, when they sign up in the system.

Each one has advantages and disadvantages. With the first one the system has problems with new users because it does not have any information about them. The second has the problems that the preferences of the user can change along the time and the information that the system stores about the user does not match with the real taste. The best solution is a mix of both techniques, using explicit information given by the users when they sign up, and updating this information along the time with the implicit information inside the ratings, using data mining techniques.

We can represent the same idea in a more formal way, following the pattern presented in [AT05]:

- *Content (i)* let be the characterization of a item *i*. It represents a set of attributes of the item.
- *ContentBasedProfile (u)* let be the profile of a user. It will be based in the historical information of the system and in the information given by the user about his or her preferences.
- *U (u, i)* is the utility function that usually represents some heuristic score.

U(u, i) = score (ContentBasedProfile (u), Content (i))

For the utility function we can use as example the cosine similarity measure:

$$\mathrm{U(u,i)} = \cos(\overrightarrow{w_u}, \overrightarrow{w_i}) = \frac{\overrightarrow{w_u} \cdot \overrightarrow{w_i}}{\|\overrightarrow{w_u}\|_2 \times \|\overrightarrow{w_i}\|_2} = \frac{\sum_{j=1}^{K} w_{j,u} w_{j,i}}{\sqrt{\sum_{j=1}^{K} w_{j,u}^2}\ \sqrt{\sum_{j=1}^{K} w_{j,i}^2}}$$

Formula 1. Utility function based in cosine similarity measure.

Where:

- $W_{j,u}$ is the weight of each attribute for each user.
- $W_{j,i}$ is the weight of each attribute for each item.

Example 1. ***Content based methods***

Assume a Films Recommender System. The system has stored the preferences of the users about films (importance of each attribute), from the information they give when they sign up in the system.

USER	ADVENTURE	COMEDY	DRAMA	CRIME	HORROR	LOVE
Elena (E)	0,20	0,20	0,10	0,10	0,05	0,25
Olalla (O)	0,10	0,10	0,30	0,25	0,10	0,15
Bruno (B)	0,20	0,30	0,10	0,15	0,20	0,05
Michael (M)	0,25	0,20	0,15	0,25	0,10	0,05

For each new film in the system is qualified using the following characteristics. Now we do not study how this information is obtained.

FILM	ADVENTURE	COMEDY	DRAMA	CRIME	HORROR	LOVE
Avatar (A)	0,35	0,10	0,05	0,20	0,05	0,25
Cars (C)	0,35	0,5	0,05	0	0	0,1
Scream (S)	0,25	0,05	0,10	0,30	0,30	0,00
Gladiator (G)	0,30	0,05	0,20	0,15	0,10	0,20

If the system uses the cosine similarity measure as utility function, below is shown how the system calculates the utility of Avatar (A) for the user Elena (E).

$$\mathrm{U(E,A)} = \frac{(0{,}2 \cdot 0{,}35) + (0{,}2 \cdot 0{,}1) + (0{,}1 \cdot 0{,}05) + (0{,}1 \cdot 0{,}2) + (0{,}05 \cdot 0{,}05) + (0{,}25 \cdot 0{,}25)}{\sqrt{0{,}2^2 + 0{,}2^2 + 0{,}1^2 + 0{,}1^2 + 0{,}05^2 + 0{,}25^2} \cdot \sqrt{0{,}35^2 + 0{,}10^2 + 0{,}05^2 + 0{,}2^2 + 0{,}05^2 + 0{,}25^2}} = 0{,}902$$

All the values of the utility function for each user/item are:

	Elena (E)	Olalla (O)	Bruno (B)	Michael (M)
Avatar (A)	**0,902**	**0,882**	0,515	**0,810**
Cars (C)	0,413	0,223	0,485	0,640
Scream (S)	0,659	**0,961**	0,692	0,587
Gladiator (G)	**0,880**	0,787	**0,826**	**0,852**

The system needs a threshold value to determine if one item can be useful for a user. This value can be defined directly in the system or it can be a parameter that the user can modify. For example in this case we suppose that the system defines 0,8 as the minimum value of the utility function to recommend the item to a specific user: only the items with bold values will be the recommend to the users.

This paradigm has the advantage of being able to recommend previously unrated items to users with unique interests and to provide explanations for its recommendations. Nevertheless, the content-based methods cannot evaluate subjective features, such as the quality of a document or a movie. Also, this technique requires analyzing the content of the items and the categorization of them this is not possible with all types of items. In other cases it is possible, but it is difficult or incurs a big computational cost.

4.2. Collaborative filtering.

The collaborative filtering is a technique for recommender systems that generates recommendations using the preferences and tastes given by others users of the system. This technique tries to simulate the collaboration in the real world between users that share opinions about recommendations and reviews [BM05].

In many cases, people have to choose between different alternatives without a complete knowledge of them. In these cases, the people believe in the recommendation of other familiar people or people whose opinion is valued by them.

The collaborative filtering systems use this idea, trying to get the users of the system that have the best opinion about an item for a user (based in his or her taste) and calculate the utility of the items for the specific user, using the opinion of the other users.

In [AT05] is explained more formally the definition of collaborative filtering. The utility *U (u, i)* that an item *s* gives to a user *c* can be calculated using the utility that the similar users $u_j \in C$ give to the same item.

4.2.1. Appropriate scenarios to use Collaborative Filtering.

The collaborative filtering can be applied in many domains, but to work properly it is better to apply it in scenarios with some characteristics [SFHS97].

The most important is that the evaluation of the items is based on subjective criteria (e.g. films) or when the items have a lot of objective criteria and they need a subjective weight to choose between them (e.g. computers). In these situations, the collaborative filtering is very powerful, but if the recommendations are only based on objective criteria, the collaborative filtering does not make sense. It does not mean that the items have no objectives criteria to be evaluated. It means that the objective criteria are very similar, and they differ only in subjective criteria.

It is important too, that each user can find others users with similar tastes, because the rating of these similar users will be an important component of recommendations. The taste of the user cannot change a lot in short time, because then the previous ratings of this user do not represent his or her preferences and they become useless for predictions.

And about the data that is necessary in one scenario to use a Collaborative Filtering system, it is important that there are many ratings per item, to ensure a good inference of recommendation and predictions and that each user rates multiple items. The system needs enough information to provide recommendations with high quality.

4.2.2. History.

The first collaborative filtering systems were explicit recommendation routed from human to human [KNNDK08]. Later, Automated Collaborative Filtering systems started to use the rating of user, to make recommendations. The opinions of user with similar taste in the past had more importance than others opinions.

This supposed an evolution of the systems, from explicit recommendations to a model of mining user preferences as the way to generate recommendations. Nowadays, many algorithms have been developed around this idea.

4.2.3. Collaborative Filtering Algorithms.

There are many possibilities to classify the collaborative filtering algorithms. In this paper, we use an organization based in [BHK98], commonly used. It distinguishes many types:

- Memory-based algorithms, that use all the ratings stored in the database to make the predictions.
- Model-based algorithms, that create a model used to calculate the predictions.
- Hybrid recommenders, those mix collaborative filtering with content based methods.

Here is a resume of the main characteristics of each one [SK09].

CF Categories	Representative Techniques	Advantages	Disadvantages
Memory-based	• Neighbour-based CF (item-based/user-based CF with Pearson/vector cosine relation) • Item-based/user-based top-N recommendations	• Easy implementation • New data can be added easily and incrementally • Need not consider the content of the items being recommended • Scale well with co-rated items	• Are dependent on human ratings • Performance decreases when data are sparse • Cannot recommend for new users and items • Have limited scalability for large datasets
Model-based	• Bayesian belief nets CF • Clustering CF • Latent Semantics Models CF • Association Rule mining	• Better address the sparsity, scalability and other problems. • Improve prediction performance • Give an intuitive rationale for recommendations	• Expensive model-building • Have trade-off between prediction performance and scalability • Lose useful information for dimensionality reduction techniques
Hybrid methods	• Content-based CF recommender • Content-booted CF • Hybrid CF combining memory-based and model-based CF	• Overcome limitations of CF and content-based or other recommenders • Improve prediction performance • Overcome CF problems such as sparsity & gray shape	• Have increased complexity and expense for implementation • Need external information that usually is not available

Table1. Types Collaborative Filtering: Techniques, advantages & disadvantages.

Before enter to explain the different collaborative filtering algorithms, we define some concepts for a good understanding:

- *Active user (U)*, as the user we try to predict the votes for.
- *Neighbour (N)*, as the users whose ratings have high correlation with the ratings of the active user.

4.2.3.1 Memory-based Algorithms.

In general, these methods use the votes of the users directly to predict the rating of the active user. One possibility is to use all the users, but usually these algorithms use only the neighbourhood of the active user (the set of user with taste most similar to the user). It is named "*User-Based Nearest Neighbour Algorithm*" and it is the most common used.

It will be more important the votes of the neighbour with most similarity with the user, so it is necessary to give weights to the ratings of the neighbours. One first idea can be a formula like the following:

$$pred(u,i) = \sum_{n \subset neighbours} userSim(u,n) \cdot r_{ni}$$

Formula 2. Basic rating estimation formula.

The result is the predicted rating value for the active user (U) for the item (I). The value is obtained as an average of the ratings of each neighbour for the item I (r_{ni}). To have a better estimation, each rating is weighted with the similarity between the user and the neighbour (userSim). Neighbours with similar taste to the user will have a higher value of userSim and his or her rating will be more considered than other neighbours that do not have anything in common with the user.

This formula has many problems like:

- If the similarities of the neighbours do not sum up to one, the prediction is incorrectly scaled. To calculate a weighted average of ratings, the sum of the coefficients that multiply the ratings has to be 1. If it is not the case, we obtain a wrong value:
  - If the sum of the similarities of the neighbours is higher than one, the predicted value will be higher than the real weighted average.
  - If the sum of the similarities of the neighbours is lower than one, the predicted value will be lower than the real weighted average.
- Different users use different rating scales in their minds. Is necessary to normalize the different scales. A rating of 7 over 10 for one user can be a good rating, but maybe if the system recommends another user the film with an estimated rating of 7 over 10, the user can think is not a good item. Is necessary to find a way to resolve the differences between the different users's rating scales.

The following formula represents the same idea, but it resolves the problems detected in the previous one:

$$pred(u,i) = \bar{r}_u + \frac{\sum_{n \subset neig\,hbours\ (u)} userSim(u,n) \cdot (r_{ni} - \bar{r}_n)}{\sum_{n \subset neig\,hbours\ (u)} userSim(u,n)}$$

Formula 3. Advanced rating estimation formula.

Where:

- u, is the user.
- n, the neighbour to compare.
- i, is a item.
- r_{ui} is the rating of the neighbour n to the item i.
- r_{ni} is the rating of the neighbour n to the item i.
- $\overline{r_n}$ is the average of rating of the neighbour.
- $\overline{r_u}$ is the average of rating of the user.

In this formula the same concept is represented as in the previous one, but in this case, for each rating the average rating of the user is subtracted, to try to normalize the scale of the different users. Also, to resolve the problem of the wrong scale due to the sum of the *userSim* coefficients not being1, the result is divided by the sum of all coefficients, normalizing the scale. Finally, to get the correct estimation the average value of the ratings of the user is added, because it was subtracted during the calculation.

To calculate the weight of each neighbour is used the similarity between the active user and the neighbour. In this case, the most common method to calculate the similarity between users is the (Pearson) correlation. It appears for first time on recommended systems in the published literature in the context of the Grouplens project [RISBR94].

$$userSim(u,n) = \frac{\sum_{i \subset CR_{u,n}} (r_{ui} - \overline{r_u})(r_{ni} - \overline{r_n})}{\sqrt{\sum_{i \subset CR_{u,n}} (r_{ui} - \overline{r_u})^2} \sqrt{\sum_{i \subset CR_{u,n}} (r_{ni} - \overline{r_n})^2}}$$

Formula 4. User similarity formula using the Pearson Correlation.

Where:

- $CR_{u,n}$ denotes the set of co-rated items between u and n.

In general, the Pearson Correlation is a measure of the correlation of two variables. In this case, these two variables are the ratings of the two users that are compared. As in the previous formula of rating estimation, all the ratings subtract the average rating of the user to prevent problems with the difference of scaling between users.

Pearson correlation returns a value in the range -1 for perfect disagreement to 1 for perfect agreement between users. The use of negative correlations to increase the prediction accuracy will be one of the aspects to analyse.

Example 2. *Memory-based Algorithm. Similarity with Pearson Correlation*

Assume a Film Recommender System that works with Collaborative Filtering. We have the information about different films rated by the users of the system. The rating is from 1 (if the user does not like at all) to 10 (if the user loves the film).

USER	Elena	Olalla	Bruno	Michael
Avatar	8	-	5	8
Cars	-	2	5	-
Scream	6	9	7	6
Gladiator	8	7	-	9
Invictus	-	-	8	9
Casablanca	5	7	-	3
The Pianist	-	8	2	4
The Godfather	8	8	6	-
The Matrix	4	2	7	-
AVERAGE	6,50	6,14	5,71	6,50

First of all, we have to calculate the user similarity with the Pearson Correlation formula. The expression below shows how is calculated the value of similarity between the users Olalla (O) and Michael (M):

$$userSim(O,M)$$

$$= \frac{(9-6{,}14)(6-6{,}5)+(7-6{,}14)(9-6{,}5)+(7-6{,}14)(3-6{,}5)+(8-6{,}14)(4-6{,}5)}{\sqrt{(9-6{,}14)^2+(7-6{,}14)^2+(7-6{,}14)^2+(8-6{,}14)^2}\sqrt{(6-6{,}5)^2+(9-6{,}5)^2+(3-6{,}5)^2+(4-6{,}5)^2}}$$

$$= -0{,}3831$$

We have the following user similarities table:

	Elena	Olalla	Bruno	Michael
Elena	1,0000	0,5852	-0,6872	0,9485
Olalla	0,5852	1,0000	-0,1712	-0,3831
Bruno	-0,6872	-0,1712	1,0000	0,7453
Michael	0,9485	-0,3831	0,7453	1,0000

Now we can calculate the predict the values of a item for a specific user with the formula 3 we showed before. Here is an example of the calculation for the user Elena (E) and the film Invictus (I). In this case we are going to consider the negative similarities as valid similarities to increasing the prediction accuracy:

$$pred(E,I) = 6{,}5 + \frac{(8-5{,}71)\cdot(-0{,}6872)+(9-6{,}5)\cdot(0{,}9485)}{(-0{,}6872)+(0{,}9485)} = 6{,}93$$

The completely table of expected ratings values for all the users is the following:

USER	Elena	Olalla	Bruno	Michael
Avatar	8,00	**6,72**	5,00	8,00
Cars	**5,45**	2,00	5,00	**6,96**
Scream	6,00	9,00	7,00	6,00
Gladiator	8,00	7,00	**6,49**	9,00
Invictus	**6,93**	**5,43**	8,00	9,00
Casablanca	5,00	7,00	**3,77**	3,00
The Pianist	7,19	8,00	2,00	4,00
The Godfather	8,00	8,00	6,00	**6,90**
The Matrix	4,00	2,00	7,00	**6,58**

The user can set a threshold that indicates the value of the expected rating value over that, it become in a recommendation. If the user *Elena* stabilised a threshold of 0,6 the system will recommends her the movie *Invictus*, but not the movie *Cars*.

With this model, we can obtain good recommendations, but there are many problems that we can try to resolve to get better recommendations. These problems are [SFHS97]:

1. Two users agreement about a universally loved item is much less important than agreement for a controversial movie. A user with few ratings who rates a universally loved item, will have a high similarity with a lot of users, because it is an item rated positively in general, but maybe the user does not have a high similarity with all their users. But if two users rate with similar values a controversial item usually it is because they have common taste.
2. The similarities with other users in the past are less important than similarities with users now. The preferences change along the time and maybe two users were similar in the past, but with the years, both change the taste in opposite ways slowly and now they have totally different preferences so they have different similarity than they had in the past.
3. With a lot of users (something essential to get good recommendations) to calculate the user's perfect neighbourhood is expensive because it requires the comparison against all other users. When there are 5.000 users it is necessary to compare each one against the other 4999. Due to, the similarity between users are symmetric, it is necessary to calculate close to 12.500.000 similarities (the half of a matrix 5.000x5.000 less the main diagonal). And each time a user rates an item is necessary to recalculate the similarity of him or her against the other 4.999 users.
4. We cannot study the similarity when there are a new users because the new users do not have ratings that let the system apply the previous similarity formulas. As we said before, the system needs enough information to provide recommendations with high quality.

Other kind of memory-based algorithms is the "*Item-Based Nearest Neighbour Algorithm*". It uses the same idea of the Used-Based Nearest Neighbour Algorithm, but in this case to calculate the predicted value of an item for a user, the system uses the ratings for the users for similar items. To know if two items are similar, it is checked if the rating from a user is similar in both items. This process is repeated for all the users that rate both items.

4.2.3.2 Model-Based Methods.

In the case of model-based methods, the system does not directly use the ratings to generate recommendations. With the ratings it creates a model to make predictions [AT05]. The models allows the system, with a previous proper training data, the recognition of patterns.

These models are developed under the formalism of different methods such as Bayesian Networks, Clustering, Association Rule Mining or Latent Semantic. All of them try to get a model that reduces the data to compute for each recommendation. There are methods that come from other fields such as machine learning and data mining.

For example, a probabilistic model proposed by [BHK98] calculates the expected value of a vote with the information of the user. If we consider votes range from 0 to m:

$$p_{u,i} = E(v_{u,i}) = \sum_{j=0}^{m} \Pr\left(v_{u,i} = j | v_{u,k}, k \epsilon I_u\right) \cdot j$$

Formula 5. Expected value of vote. Probabilistic model.

For each one of the possible values to rate (from 0 to m), is calculated the probability that the user U give the value j under the know condition of the rating that the user gives ($v_{u,k}$) to others items (I_u). Each probability value is multiply by the rating value (j). The idea is a weighted average of the possible values of the rating, proportional to the probably of each value of the rating.

Many methods have been studied to calculate this value [SK09].

- *Bayesian Networks Models*: they have a node for each item and the states of each node are the possible vote values that they can give. There is another value to represent "no-vote" for missing data. At the end, in the resulting network, each item will have a set of parent items that are the best predictors of its votes. Each conditional probability table is represented by a decision tree.
- *Cluster Models:* Clustering technique partitions the objects in groups (named clusters) that are similar to one another within the same cluster and are dissimilar to the objects in other clusters. To measure the similarity between objects is used metrics such as Minkowski distance and Pearson correlation. Clustering methods can be classified in:

    - Partitioning methods (e.g. k-means), that select one object as centroid of each cluster and they create the clusters using this measure of distance from the objects to the centroids.
    - Density-based methods (e.g. DBSCAN, OPTICS) search for dense clusters of objects separated by sparse regions (the noise).
    - Hierarchical methods (e.g. BIRCH), use some rules to create hierarchical decomposition of the set of data objects.
- *Latent Semantic Models:* is a dimensionality reduction technique which is widely used in information retrieval. Each dimension in the reduced space is a latent variable (or factor) representing groups of highly correlated index terms. Reducing the dimensionality of the original matrix reduces the amount of noise in the data, improving retrieval based on the computation of similarities.
- *Association rule mining:* is a technique to find correlations between the items in a dataset [CR06]. It tries to obtain rules from the dataset that derive new knowledge. The rules set conditions (about taste or profile of the user) that if matched with the active user, the system recommends a specific item to the user. It is a technique that generates an easy to understand representation for the users. The association mining analysis is a two part process:
    - The identification of frequents sets of items in the dataset.
    - The derivation of inferences from the set of items detected.

4.2.3.3 Hybrid methods.

The hybrid methods try to get the advantages of the collaborative filtering and the content-based methods combining both. There are different ways to mix them:

1. The system uses both methods separately and then it gets the recommendation as the result of combining them.
2. The system implements a collaborative filtering approach but adds some content-based features to improve the quality of the recommendations.
3. The system implements a content-based approach but add some collaborative filtering features to improve the quality of the recommendations.
4. The system implements a model where both approach are completely unified.

Chapter 5. Improving the model.

After an overview of the most of the possibilities, we decide to work around a recommender system which uses collaborative filtering to generate recommendations and uses the User-Based Nearest Neighbour Algorithm.

The reason for choosing this algorithm is that gives facilities to test the influence of the different similarity formulas directly in the recommendations. We will study the quality of the similarity formulas evaluating the recommendation that the system predicts with them.

One of the problems that the user similarity has using the Pearson correlation is to distinguish similarities when the common ratings are universally loved item from controversial movies. To try to resolve this problem, the variance in the item rating can be included to calculate the user similarity. The controversial items usually will have more variance than the universally loved items. We can know the average variance in the opinions of the users when they rate films, and we can know the variance of the rating of a specific item.

$$userSim(u,n) = \frac{\sum_{i \subset CR_{u,n}} (r_{ui} - \overline{r_u})(r_{ni} - \overline{r_n})\,(Var(i)/\overline{Var})}{\sqrt{\sum_{i \subset CR_{u,n}} (r_{ui} - \overline{r_u})^2\,(Var(i)/\overline{Var})}\,\sqrt{\sum_{i \subset CR_{u,n}} (r_{ni} - \overline{r_n})^2 (Var(i)/\overline{Var})}}$$

Formula 6. User similarity formula adding variance.

In this way, similarities in films with more variety of opinions are more valued than similarities in films very popular with a lot of similar opinions.

In [BHK98] exposes another idea: under the initial algorithm there is no role for negative votes, and unobserved items receive a zero vote.

They propose the following user similarity calculation:

$$userSim(u,n) = \sum_{i \subset CR_{u,n}} \frac{r_{ui} \cdot r_{ni}}{\sqrt{\sum_{j \subset CR_u} {r_{ui}}^2}\,\sqrt{\sum_{j \subset CR_n} {r_{ni}}^2}}$$

Formula 7. User similarity using all ratings.

The squared terms in the denominator serve to normalize votes so users that vote on more items will not be a priori more similar to other users.

The last formula proposed joins the two previous solutions. Over the idea of the formula 7 using all the ratings, it is added in each sum the influence of the variance of the ratings of each item, as in the solution proposed in the formula 6.

$$userSim(u,n) = \sum_{i \subset CR_{u,n}} \frac{r_{ui} \cdot r_{ni} \cdot (Var(i)/\overline{Var})}{\sqrt{\sum_{j \subset CR_u} {r_{ui}}^2 \, (Var(j)/\overline{Var})} \sqrt{\sum_{j \subset CR_n} {r_{ni}}^2 \, (Var(j)/\overline{Var})}}$$

Formula 8. User similarity formula using all ratings and variance.

Here is presented a simple example of how the previous formulas work. The quality of the similarity values a predicted recommendations is discussed in the chapter "Test of the similarities formulas".

Example 3. ***Collaborative Filtering. Improving Similarity formulas***

We have the information about different films rating by the users of the system. The rating is from 1 (if the user doesn't like at all) to 10 (if the user love the film).

USER	Elena	Olalla	Bruno	Michael
Avatar	8	-	5	8
Cars	-	2	5	-
Scream	6	9	7	6
Gladiator	8	7	-	9
Invictus	-	-	8	9
Casablanca	5	7	-	3
The Pianist	-	8	2	4
The Godfather	8	8	6	-
The Matrix	4	2	7	-

If we introduce the variance factor over the Pearson Correlation:

$$userSim(u,n) = \sum_{i \subset CR_{u,n}} \frac{(r_{ui} - \overline{r_u})(r_{ni} - \overline{r_n})(Var(i)/\overline{Var})}{\sqrt{\sum_{j \subset CR_u} {r_{ui}}^2 \, (Var(j)/\overline{Var})} \sqrt{\sum_{j \subset CR_n} {r_{ni}}^2 \, (Var(j)/\overline{Var})}}$$

The similarities table is:

	Elena	Olalla	Bruno	Michael
Elena	1,0000	0,7254	-0,8699	0,9409
Olalla	0,7254	1,0000	-0,4417	-0,7169
Bruno	-0,8699	-0,4417	1,0000	0,8752
Michael	0,9409	-0,7169	0,8752	1,0000

If we use the user similarity formula 6:

$$userSim(u,n) = \sum\nolimits_{i \subset CR_{u,n}} \frac{r_{ui} \cdot r_{ni}}{\sqrt{\sum_{j \subset CR_u} {r_{ui}}^2}\ \sqrt{\sum_{j \subset CR_n} {r_{ni}}^2}}$$

We obtain the following user similarities table:

	Elena	Olalla	Bruno	Michael
Elena	1,0000	0,7455	0,6068	0,6730
Olalla	0,7455	1,0000	0,5359	0,5654
Bruno	0,6068	0,5359	1,0000	0,6024
Michael	0,6730	0,5654	0,6024	1,0000

And if we use the calculation of the user similarity formula 7:

$$userSim(u,n) = \sum\nolimits_{i \subset CR_{u,n}} \frac{r_{ui} \cdot r_{ni} \cdot (Var(i)/\overline{Var})}{\sqrt{\sum_{j \subset CR_u} {r_{ui}}^2\,(Var(j)/\overline{Var})}\ \sqrt{\sum_{j \subset CR_n} {r_{ni}}^2\,(Var(j)/\overline{Var})}}$$

The table is the following:

	Elena	Olalla	Bruno	Michael
Elena	1,0000	0,5358	0,6952	0,6794
Olalla	0,5358	1,0000	0,5366	0,7011
Bruno	0,6952	0,5366	1,0000	0,5045
Michael	0,6794	0,7011	0,5045	1,0000

With the similarity tables is easy calculate the predicted value of recommendations. In the next example are used the similarity table of the previous one to recommend items to the users. As in the previous example, here only is explained how the system make the recommendation but ther is not discussion about the quality of the recommendations, because it is be analyze in the chapter “Test of the similarities formulas”.

Example 4. ***Collaborative Filtering. Predicted values.***

Now we can calculate the predict the values of a item for a specific user with the formula:

$$pred(u,i) = \bar{r}_u + \frac{\sum_{n \subset neighbours\ (u)} userSim(u,n) \cdot (r_{ni} - \bar{r}_n)}{\sum_{n \subset neighbours\ (u)} userSim(u,n)}$$

With user similarities formula 6:

USER	Elena	Olalla	Bruno	Michael
Avatar	8,00	**6,72**	5,00	8,00
Cars	**5,17**	2,00	5,00	**7,62**
Scream	6,00	9,00	7,00	6,00
Gladiator	8,00	7,00	**6,61**	9,00
Invictus	**6,70**	**1,20**	8,00	9,00
Casablanca	5,00	7,00	**1,92**	3,00
The Pianist	**7,74**	8,00	2,00	4,00
The Godfather	8,00	8,00	6,00	**6,66**
The Matrix	4,00	2,00	7,00	**7,33**

With user similarities formula 7:

USER	Elena	Olalla	Bruno	Michael
Avatar	8,00	***10,77***	5,00	8,00
Cars	***8,00***	2,00	5,00	***7,96***
Scream	6,00	9,00	7,00	6,00
Gladiator	8,00	7,00	***10,82***	9,00
Invictus	***10,11***	***9,44***	8,00	9,00
Casablanca	5,00	7,00	***8,84***	3,00
The Pianist	***9,76***	8,00	2,00	4,00
The Godfather	8,00	8,00	6,00	***11,26***
The Matrix	4,00	2,00	7,00	***9,33***

With user similarities formula 8:

USER	Elena	Olalla	Bruno	Michael
Avatar	8,00	***10,68***	5,00	8,00
Cars	***8,06***	2,00	5,00	***7,86***
Scream	6,00	9,00	7,00	6,00
Gladiator	8,00	7,00	***10,78***	9,00
Invictus	***10,51***	***9,97***	8,00	9,00
Casablanca	5,00	7,00	***8,91***	3,00
The Pianist	***9,38***	8,00	2,00	4,00
The Godfather	8,00	8,00	6,00	***11,38***
The Matrix	4,00	2,00	7,00	***9,15***

We can fix a threshold for the estimated rating over it we can recommend the item to the user. If we set a threshold of 0.7, we can recommend:

- To Elena watch “The Pianist”.
- To Michael watch “Cars” and “The Matrix”.

Chapter 6. Giving feedback.

When the system generates recommendations for a user, they have to be presented in a easily understandable way that non expert users can understand the model.

This presentation has to show a list of similar (anonymous) users and a list of recommended items for the active user.

In the moment the recommendation model is shown to the user, he or she can decide to change the rules that the model is using to calculate the recommendations. The system has to learn from the feedback the user gives.

The most common situations can be:

1. The active user decides that his taste is more similar to another specific user, than the similarity calculated by system.
2. The active user decides that his taste is less similar to another specific user, than the similarity calculated by system.
3. The active user does not want that a specific item be used by the system to make predictions and to calculate similarities between the active user and other users.
4. The active user does not want that a specific user be used by the system to make predictions for him or her.

6.1. The active user sets similarity manually.

The system calculates the similarities of the active user with his or her neighbours and it shows a list of the users similar to the active user.

However, the active user may decide that his or her taste really is more (or less) similar to the taste of other users, than the similarity calculated by the system between them.

One solution to give this feedback to the system is offering the user to set another rating for the similarity with each user. It is a similarity enhancer with a range of values positives and negatives and with a default value of 0. When the system calculates the similarity between two users (active user and neighbour), it will multiply the similarity value estimated by the following coefficient:

$$coeficient = 1 + \frac{EnhancerValue}{10}$$

Formula 9. Similarity enhancer coefficient

Using this coefficient:

- If the similarity enhancer is 0, the coefficient is 1 and the similarity does not change.
- If the similarity enhancer takes positive values, the coefficient will have a value higher than one, so the similarity will increase.
- If the similarity enhancer takes negative values, the coefficient will have a value lower than one, so the similarity will decrease.

Here is a practical example that shows how the similarity enhancer works.

Example 5. ***Collaborative Filtering. Similarity enhancer.***

We are going to use the similarities formula 4, that it gave the following similarity table:

	Elena	Olalla (User1)	Bruno (User2)	Michael (User3)
Elena	1,0000	**0,5358**	**0,6952**	**0,6794**
Olalla	0,5358	1,0000	0,5366	0,7011
Bruno	0,6952	0,5366	1,0000	0,5045
Michael	0,6794	0,7011	0,5045	1,0000

With this information the system shows to the user Elena that she has similarity with 3 users. The scale is transforming from [0 .. 1] to [0% .. 100%].

Top 3 similarity Elena	
User 2	69,52%
User 3	67,94%
User 1	53,58%

But the user Elena often uses the system and explores the profiles of other people. She thinks her similarity with these 3 users does not correspond with the similarity estimated by the system. She considers her similarity with user 3 is much less than the one given by the system and the similarity with user 1 is a bit higher than what the table shows.

Elena similarity enhancer	
User3	-5
User1	+2
User2	0

In this case, the system will adapt the similarity calculation:

- For User2, the calculation will be the same.
- For User3, the system will use a factor to correct the similarity. The similarity calculated by the system is multiplied by a factor of 0,5 (the similarity enhancer absolute value divided 10).
- For User1, the system has to increase the similarity. The similarity calculated by the system is multiplied by a factor of 1,2 (one more than the similarity enhancer absolute value divided 10).

So, the new similarities that the system will use to calculate the recommendations is:

Top 3 similarity Elena	
User 2	69,52%
User 1	64,30%
User 3	33,97%

In this case the new similarity between users is:

	Elena	Olalla (User1)	Bruno (User2)	Michael (User3)
Elena	1,0000	**0,6430**	**0,6952**	**0,3397**
Olalla	0,5358	1,0000	0,5366	0,7011
Bruno	0,6952	0,5366	1,0000	0,5045
Michael	0,6794	0,7011	0,5045	1,0000

We can see that now is not a symmetric matrix. Although, Elena decided to change the similarity with Michael, Michael does not make any change. Is the same for Olalla. Now, to make recommendation to Elena, Olalla is so much important than Michael, while before Michael was more important.

USER	Elena (Before)	Elena (After)	Olalla (User1)	Michael (User3)
Avatar	8,00	8,00	7,33	8,00
Cars	**5,57**	**5,32**	**2,00**	5,37
Scream	6,00	6,00	9,00	6,00
Gladiator	8,00	8,00	7,00	9,00
Invictus	**7,63**	**7,41**	8,55	**9,00**
Casablanca	5,00	5,00	7,00	3,00
The Pianist	**5,37**	**5,66**	**8,00**	**4,00**
The Godfather	8,00	8,00	8,00	***6,44***
The Matrix	4,00	4,00	2,00	***7,74***

As we can see, the recommendations for Elena after the recommendation enhancer are closer to the taste of Olalla than before. It is the opposite for Michael. The recommendations for Olalla and Michael do not change.

6.2. Skip neighbours in recommendations.

Maybe a user does not like the taste from a specific neighbour and does not want to receive recommendations influences by this neighbour.

In cases like this, the system has to provide a mechanism that allows the active user to hide a specific neighbour for the calculations. It will be necessary to have a list of the users that the active user does not want to use in his or her calculations.

Example 6. ***Collaborative Filtering. Skip neighbours in recommendations.***

If we continue with the similarities of the formula 4:

	Elena	Olalla (User 1)	Bruno (User 2)	Michael (User 3)
Elena	1,0000	0,5358	0,6952	0,6794
Olalla	0,5358	1,0000	0,5366	0,7011
Bruno	0,6952	0,5366	1,0000	0,5045
Michael	0,6794	0,7011	0,5045	1,0000

When the system shows the Top of similarities, the user Elena can choose to delete some users. They will not be used to make the calculation of the recommendations. In the following example, the user chooses User3 will not be used for the calculation

Top 3 similarity Elena		
User 2	69,5%	✗
User 3	67,9%	
User 1	53,6%	

In this case, the new similarity table will be:

	Elena	**Olalla (User 1)**	**Bruno (User 2)**	**Michael (User 3)**
Elena	1,0000	0,5358	**0,0000**	0,6794
Olalla	0,5358	1,0000	0,5366	0,7011
Bruno	0,6952	0,5366	1,0000	0,5045
Michael	0,6794	0,7011	0,5045	1,0000

With a correlation between users of “0”, the opinion of User2 does not have any value for the user Elena. We can see how changes the recommendations table.

USER	**Elena (Before)**	**Elena (After)**	**Olalla**	**Bruno**	**Michael**
Avatar	8,00	8,00	7,33	5,00	8,00
Cars	**5,51**	**5,43**	2,00	5,00	5,30
Scream	6,00	6,00	9,00	7,00	6,00
Gladiator	8,00	8,00	7,00	6,72	9,00
Invictus	**7,61**	**7,25**	8,55	8,00	9,00
Casablanca	5,00	5,00	7,00	4,89	3,00
The Pianist	**5,42**	**6,23**	8,00	2,00	4,00
The Godfather	8,00	8,00	8,00	6,00	7,38
The Matrix	4,00	4,00	2,00	7,00	5,07

The new values of recommendation for Elena are now further from User2.

6.3. Skip items in similarity calculation.

As shown in the previous case, it may happen a user does not want a specific item to be used to make the calculation of similarity between users. Maybe, the user likes a item from a category lot, but he or she only does because that item has a special meaning for the user. He or she does not want to receive recommendations about this category of items. Thus one solution is to work without the ratings of this special item.

It will be necessary to have a list for each user with all the items that he or she does not want to be used in the calculation. There will be two cases depending the similarity formula that uses the system:

- With the formulas 1 and 2, an item is used to calculate the similarity only in the case that the user rates the item. In these cases:
    - If it is an item unrated by the user, the system keeps the item in the "skipped items list" because maybe in the future the user rates the item and is necessary to know that the system does not have to use to make calculations.
    - If it is an item rated by the user, the system adds the item to the "skipped items list" and recalculates the similarity for the user and the recommendations.
- With the formulas 3 and 4, an item is used to calculate the similarity in all the cases, so the system always has to add the item to the "skipped items list" and recalculates the similarity and recommendations for the user.

The average of votes for each user and the average variance are values calculated for all the items before calculating the similarities. Their recalculation for all the users who have skipped items needs long time for computing . The system will estimate the average rating of the user and the average variance each time a skipped item in the calculations appears. The details of this estimation are discussed in the design section.

Example 7. ***Collaborative Filtering. Skip items in similarity calculation.***

The system lets users explore all the items, for rating or only to get information about them. When a user explores an item, he or she can decide that the system does not use the specific item to make the calculations of similarity. We suppose the user Elena does not want to use the item "Casablanca" for calculations, because it is a movie very different from her personal taste and she is aware it is a negative influence in her recommendations.

Casablanca

Director:	Michael Curtiz
Rated with:	5
Number of ratings:	3
Relevant users that votes:	Me, User1, User3

When the system receives this feedback it, has to recalculate the similarity for the user Elena. The new similarity table is the following:

	Elena	Olalla	Bruno	Michael
Elena	1,0000	**0,4438**	**0,7587**	**0,6526**
Olalla	0,5358	1,0000	0,5366	0,7011
Bruno	0,6952	0,5366	1,0000	0,5045
Michael	0,6794	0,7011	0,5045	1,0000

USER	Elena (before)	Elena (after)	Olalla	Bruno	Michael
Avatar	8,00	8,00	7,33	5,00	8,00
Cars	**5,51**	**5,67**	2,00	5,00	5,30
Scream	6,00	6,00	9,00	7,00	6,00
Gladiator	8,00	8,00	7,00	6,72	9,00
Invictus	**7,61**	**7,68**	8,55	8,00	9,00
Casablanca	5,00	5,00	7,00	4,89	3,00
The Pianist	**5,42**	**5,23**	8,00	2,00	4,00
The Godfather	8,00	8,00	8,00	6,00	7,38
The Matrix	4,00	4,00	2,00	7,00	5,07

We can see that the item has influence in the similarities of the user with the neighbours and as consequence in the predicted values of the recommendations.

Chapter 7. Evaluation metrics.

The evaluation metrics let study how well the system is predicting the ratings. They give a measure of the difference between the predicted rating of a user for an item, and the real value that the user rates this item with [HKTR04].

In Collaborative Filtering exists two main ways to show e recommendations to the users [BHK98].

- The first one shows recommendation to the user individually, with the predicted value of the rating for the specific user. In this case, each item recommended to each user will have associated a predicted value. This is the case of systems like MovieLens[1].
- The other kind of recommender systems show to the user a list of the recommended items ordered from the highest ranked to the lowest. It is supposed that the user will explore the items in the order than appear in the list, from the top. This is the base idea of many Internet Search Engines (like the recommended videos in YouTube[2])

7.1. Metrics to evaluate the recommendation of system.

To evaluate the system we use one metric for each type of recommendations with propose of discover in which system type match better the techniques are discussed in this document.

In both cases the procedure consists in split the user dataset in two. One named *training set* that is used as database to get the information to make predictions. The other one named *test set* that consider the content as active users using some of the ratings as observed and predicting the rest. Then, these predicted values are analyzed with the following techniques.

7.1.1. Average absolute deviation for a user.

As we explain before, using the metric, there are a dataset named test set with some items observed and other items that are predicted (even they are observed too and we have the real value assigned by the user, but is hidden for the moment).

[1] http://www.movielens.org
[2] http://www.youtube.com

For each individual score of the predicted ratings in test set, is calculated the absolute deviation between the predicted value and the real vote of the user. Then, it is calculated the average of this values for each user and finally the average value for all the users of the test set.

The formal idea is the following [BHK98]. There is an active user (A) that has a set of observed values (I_A) and a set of predicted values (P_A) that the system calculates with I_A, but is known the real value that the user rates (V_A). If the number of predicted values for the active user is m_A the average absolute deviation for a user will be:

$$S_A = \frac{1}{m_A} \sum_{j \in P_A} |p_{a,j} - v_{a,j}|$$

Formula 10. Average absolute desviation formula

7.1.2. Precision and Recall and F1.

First of all, this method distinguishes between relevant and not relevant items. In the case of binary recommendation this is not a problem. Nevertheless, when the system works with rating scales, the rating values have to be transformed in a binary scale. In order to transform them into binary scale, it is necessary to establish a threshold over which the rating is tagged as relevant and below which the rating is tagged as no relevant. The threshold value will be set depending on the scale of the ratings used in the recommender system.

As in the other metric, the data set is split in test set and training set, and the result has to be split to into the set of items returned to the user (the recommended items) and the set did not.

The metric defines *precision* as the number of (real) relevant items selected divides the total number of selected items. Represent the probability that a selected item can be relevant.

$$Precision = \frac{|I_{Relevant} \cap I_{Selected}|}{|I_{Selected}|}$$

Formula 11. Precision metric

On the other hand *recall* represents the ratio of relevant items selected from the whole set of relevant items.

$$Recall = \frac{|I_{Relevant} \cap I_{Select\ ed}|}{|I_{Relevan}|}$$

Formula 12. Recall metric

With this two measure is possible to make a complete analysis how well the recommendations of the system works. But it can be more complete if we use a measure that combines both. It is the case of the F1 measure.

F1 is a measure that combines the others previous two with an equal weight. It ranges from 0 to 1 where higher values indicate better recommendations.

$$F1 = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision}$$

Formula 13. F1 metric

7.2. Cases of study.

Already we know how to study the quality of the recommendations, but now is time to define the situations to apply the metrics. The following cases make up the test plan that is followed in test chapter.

7.2.1. General evaluation of time processing.

It is necessary to know the time of processing of the data, because this kind of system work with large amounts of data and it suppose long time for load, process and keep the data.

With this propose will be calculated the time that the system need to:

- Load the basic information for the first run. It supposes to load the information about users, items and the ratings of the users for the ratings.
- Calculate the similarity between all the users of the system (for each similarity formula).
- Load the similarities from the file.
- Calculate the recommended items for the users, when the similarities are in the system (calculated or loaded from a file).

These cases have to be run with different database sizes.

7.2.2. Evaluation of the different similarity formulas.

For each one of the similarity formulas was exposed, it will be applied the metrics showed to measure the recommendations of the system. The next scenarios will be the cases of study:

- The system has few users with few items rated by each one.

- The system has few users but with a lot of items rated by each one.
- The system has a lot of user and each one rates a lot of items.
- The system has few users all of them with similar ratings.
- The system has few users. All of them except two are have very similar ratings. These two are very different from the others, but very similar between them.
- The system has a lot of users and each one rate a lot of items, but in the recommendation are used the negative similarities.

7.2.3. Evaluation of the feedback techniques.

In the case of the feedback, it will be choose the similarity formula that best results gives and will be study the recommendations variation when it is applied feedback or when is not applied. The cases of feedback that will be test are:

- Skip one, many and a lot of items when there are a lot of items.
- Skip one, many and a lot of items when there are few of items.
- Skip one, many and a lot of users when they are very similar to the user.
- Skip one, many and a lot of users when they are not very similar to the user.
- Apply the similarity enhancer in one, many and a lot of users increasing the similarity when they are very similar to the user.
- Apply the similarity enhancer in one, many and a lot of users increasing the similarity when they are not similar to the user.
- Apply the similarity enhancer in one, many and a lot of users decreasing the similarity when they are very similar to the user.
- Apply the similarity enhancer in one, many and a lot of users decreasing the similarity when they are not similar to the user.

The aim for use one, many and a lot of users/items in the cases previous mentioned, is to study how the feedback evolves in each one of the cases proposed.

PART II

DESIGN

Chapter 8. Structure of the RS.

One of the most common structures to represent all the information in the system is a graph. Typically a graph is represented in memory as a list of nodes and edges in a matrix. The nodes represent the users and the edges represent the similarity between them.

In each node, the system stores the information relative to the user, and in each edge it stores the similarity between users. It is not symmetric, due to the “enhanced mechanism”, the “skipped items” and the “skipped users”, so there are stored the values of similarity for each of the two ways. It is an undirected graph.

Furthermore, the system stores the list of the items with all the information relative to them, even including the ratings that the users give to the items.

In the followings sections is described with more detail the structure of each one of the components of the system, and the most relevant characteristics of them.

8.1. Ratings.

The ratings the users give to the items have to be stored in some place to use them for the calculation. The information needed for each rating is: the user that gives the rating, the item evaluated and the value of the rating.

In many cases, the user objects and the item objects have to access to these values to make calculations. The user objects need to access to all the rates given by a specific user. For the items it is necessary to access to all the ratings the users gave for a specific item.

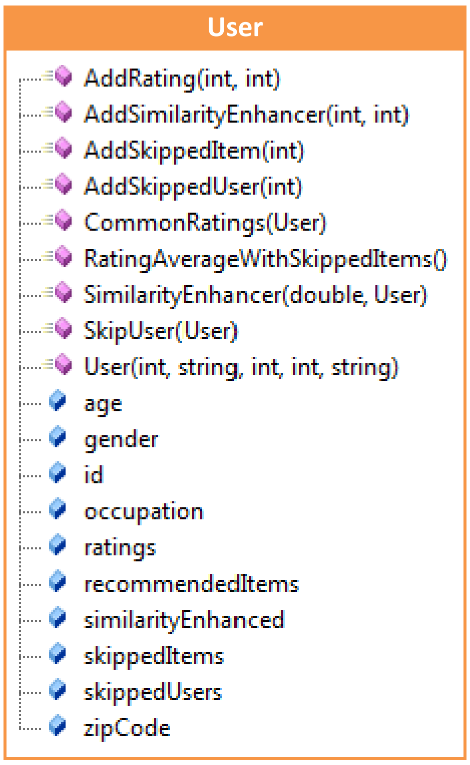
There are many ways to store and manage this information. The two main possibilities studied were:

- To create an independent data structure that stores all the ratings. The rest of classes of the system accesses to this structure when they need to make some calculation with the ratings. Due to that, it is only possible to order the ratings in memory by only one criteria (user or item). It is necessary to implement methods to access the data. As a result, an extra computing cost will be required.
- To divide the data structure in two, storing in each user all the rating given by him or her, and storing in each item all the rating that it receives. This way needs the double of memory but it ensures faster access to the ratings given to a specific item or by a specific user.

The solution the system uses is the second one. Due to this, calculating the recommendations and the similarities with large datasets will require a lot of computational time. We decide to spend memory to save it.

8.2. Users.

The component user represents all the information relative to the user such as the profile information and the values of the rating given by the user. The operations are necessary to make the calculation with the stored data. The structure of the component user is the following:



User

- AddRating(int, int)
- AddSimilarityEnhancer(int, int)
- AddSkippedItem(int)
- AddSkippedUser(int)
- CommonRatings(User)
- RatingAverageWithSkippedItems()
- SimilarityEnhancer(double, User)
- SkipUser(User)
- User(int, string, int, int, string)
- age
- gender
- id
- occupation
- ratings
- recommendedItems
- similarityEnhanced
- skippedItems
- skippedUsers
- zipCode

Graph1. Attributes and methods of the class User.

Each user component has to store the typical profile information of a user and also these ones:

- *ratings.* The list of all the rating given by the user. For each rating the item rated and the value given are stored.
- *recommendedItems*. The list with the predicted values of the items recommended to the user. For each one the item and the predicted value are stored.
- *similarityEnhancer*. The list that stores the enhancer values with the user that refers to.
- *skippedItems*. The list that stores the items to skip in the calculations.
- *skippedUsers*. The list that stores the users to skip in the calculations.

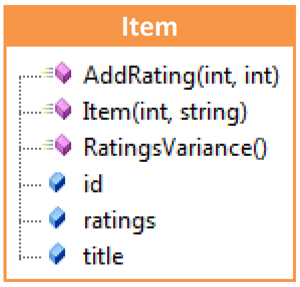
Here are the relevant operations a user needs:

- *Add methods* that insert the values in the proper list.
- *CommonRatings (User)* get the list of common ratings between the current user and the user given as parameter.

- *RatingAverageWithSkippedItems()*, return the average of the ratings of the user but does not use the skipped items in the calculation.
- *SimilarityEnhancer (double, User)*, calculate the similarity between the current user and another one, applying the value of stored similarity enhancer over the initial similarity (received as parameter). If there is no enhancer value for the user received as parameter, the method returns the initial similarity.
- *SkipUser (User)*, return Boolean indicating if the current user has to skip the rates of the user given as parameter.

8.3. Items.

The component item stores the information relative to an item such as the basic characteristics (in this case the id and title) and the values of the rating given to the item. The structure of the component item is the following:



Graph2. Attributes and methods of the class Item.

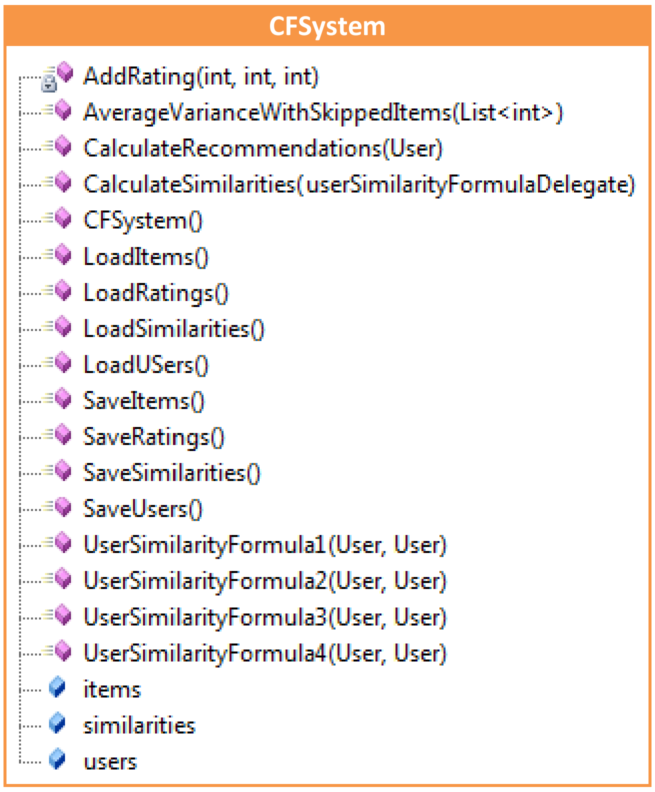
Each item component stores the information necessary for an item and the list of all the rating given to the item by the users. For each rating, the user who rates and the value given by the user are stored.

The operation *RatingVariance()* gets the variance of the ratings of the item.

8.4. Collaborative filtering system.

The component CFSystem involves all the structure of the recommender system. It has the list of users and items. It means it has the ratings (inside the users and the items). In this way, it can make the calculation of the similarities between users and store them.

The structure of the component CFSystem is the following:


CFSystem

AddRating(int, int, int)
AverageVarianceWithSkippedItems(List<int>)
CalculateRecommendations(User)
CalculateSimilarities(userSimilarityFormulaDelegate)
CFSystem()
LoadItems()
LoadRatings()
LoadSimilarities()
LoadUSers()
SaveItems()
SaveRatings()
SaveSimilarities()
SaveUsers()
UserSimilarityFormula1(User, User)
UserSimilarityFormula2(User, User)
UserSimilarityFormula3(User, User)
UserSimilarityFormula4(User, User)
items
similarities
users

Graph3. Attributes and methods of the class CFSystem.

Where:

- *items*, is a list with all the components *item* of the system.
- *users*, is a list with all the components *user* of the system.
- *similatiries*, stores all the relevant similarities to make recommendations of the users against all the users. What relevant similarity means, is explained in 8.5 Calculating the Similarities.
- *AddRating (int, int, int)*, includes a new rating value in the system. The rating has a user (that rates,) an item (that is rated) and the value. This informations has to be stored twice in the system:
  - *the id of the item and the value of the rating have to be stored in the correspondent user component, in its ratings list.*
  - *list the id of the user and the value of the rating have to be stored In the correspondent item component, in its ratings.*
- *AverageVarianceWithSkippedItems (List<int>)*, returns the average of the variance of all the items but does not use the skipped items in the calculation.
- *CalculateSimilarities (userSimilarityFormulaDelegate)*, calculate the similarities of all the users against all the users applying one of the similarity formulas that receive as parameter. All the relevant similarities are stored in the similarities attribute.
- *Load...()*, loads the respective information from a file to the system.
- *Save...()*, saves the respective information to a file.
- *UserSimilarityFormula...(User, User)*, calculates the similarity between the two users given as parameter using each one of the formulas studied in the analysis section. In the next section each formula is detailed.

8.5. Calculating the Similarities.

The four formulas used to calculate the similarity between users are the four formulas described in the analysis.

UserSimilarityFormula1 represents:

$$userSim(u,n) = \frac{\sum_{i \subset CR_{u,n}} (r_{ui} - \overline{r_u})(r_{ni} - \overline{r_n})}{\sqrt{\sum_{i \subset CR_{u,n}} (r_{ui} - \overline{r_u})^2} \sqrt{\sum_{i \subset CR_{u,n}} (r_{ni} - \overline{r_n})^2}}$$

UserSimilarityFormula2 represents:

$$userSim(u,n) = \frac{\sum_{i \subset CR_{u,n}} (r_{ui} - \overline{r_u})(r_{ni} - \overline{r_n}) \left(Var(i)/\overline{Var}\right)}{\sqrt{\sum_{i \subset CR_{u,n}} (r_{ui} - \overline{r_u})^2 \left(Var(i)/\overline{Var}\right)} \sqrt{\sum_{i \subset CR_{u,n}} (r_{ni} - \overline{r_n})^2 \left(Var(i)/\overline{Var}\right)}}$$

UserSimilarityFormula3 represents:

$$userSim(u,n) = \sum\nolimits_{i \subset CR_{u,n}} \frac{r_{ui} \cdot r_{ni}}{\sqrt{\sum_{j \subset CR_u} {r_{ui}}^2} \sqrt{\sum_{j \subset CR_n} {r_{ni}}^2}}$$

UserSimilarityFormula4 represents:

$$userSim(u,n) = \sum\nolimits_{i \subset CR_{u,n}} \frac{r_{ui} \cdot r_{ni} \cdot \left(Var(i)/\overline{Var}\right)}{\sqrt{\sum_{j \subset CR_u} {r_{ui}}^2 \left(Var(j)/\overline{Var}\right)} \sqrt{\sum_{j \subset CR_n} {r_{ni}}^2 \left(Var(j)/\overline{Var}\right)}}$$

The range of the values that these formulas return is from -1 to 1, but, as we said in the analysis chapter when the Pearson Correlation was explained, there are two ways to work with the negative similarities.

The system has both possibilities, but it works differently in each case. When the similarities between all the users are calculated, only the relevant similarities are stored in the *similarities* structure. When the system has to consider the negative values, the relevant similarities are all of them, different from 0. But when the system has to consider only the positives values, only the values higher than 0 are stored in the system. The aim is to save as much memory as possible.

Chapter 9. Test files.

9.1. Test files used.

In order to test the system correctly, it is necessary to have large datasets. It would be better if these datasets were built with real data. This task is very hard and takes a long time to recollect all the information.

Because of that, a large dataset was taken from the Internet using the research lab GroupLens[3]. More precisely, we downloaded a dataset of their recommender system of movies MovieLens. This data set has 1 million of ratings for 3952 movies by 6040 users.

They allow to use the dataset for research purposes under some conditions which were respected.

However, some of the tests are designed by our own way, because it is necessary to have some scenarios that the real data does not recreate.

9.2. Structure of the test files.

For the test we saved the original structure of the files but for the users we put the additional information for the feedback techniques.

9.2.1. Ratings file description.

All ratings are contained in the file "ratings.dat" and are in the following format:

UserID::MovieID::Rating::Timestamp

- UserIDs range between 1 and 6040
- MovieIDs range between 0 and 3592
- Ratings are made on a 5-star scale (whole-star ratings only)
- Timestamp is represented in seconds since the epoch as returned by time(2)
- SimilarityEnhancer is alist

Each user has at least 20 ratings.

[3] http://www.grouplens.org/

9.2.2. Users file description

User information is in the file "users.dat" and is in the following format:

UserID::Gender::Age::Occupation::Zip-code::SimilarityEnhancer::SkippedItems::SkippedUsers

- Gender is denoted by a "M" for male and "F" for female
- Age is chosen from the following ranges:
  - 1: "Under 18"
  - 18: "18-24"
  - 25: "25-34"
  - 35: "35-44"
  - 45: "45-49"
  - 50: "50-55"
  - 56: "56+"
- Occupation is chosen from the following choices:
  - 0: "other" or not specified
  - 1: "academic/educator"
  - 2: "artist"
  - 3: "clerical/admin"
  - 4: "college/grad student"
  - 5: "customer service"
  - 6: "doctor/health care"
  - 7: "executive/managerial"
  - 8: "farmer"
  - 9: "homemaker"
  - 10: "K-12 student"
  - 11: "lawyer"
  - 12: "programmer"
  - 13: "retired"
  - 14: "sales/marketing"
  - 15: "scientist"
  - 16: "self-employed"
  - 17: "technician/engineer"
  - 18: "tradesman/craftsman"
  - 19: "unemployed"
  - 20: "writer"
- SimilarityEnhancer is a collection of pairs of UserId and Value, like the following:

UserId=Value%UserId=Value%UserId=Value%UserId=Value%...

- SkippedItems is a collection of id of items separated by the symbol “%”.
- SkippedUsers is a collection of id of users separated by the symbol “%”.

9.2.3. Movies file description.

Movie information is in the file "movies.dat" and is in the following format:

MovieID::Title::Genres

- Titles are identical to titles provided by the IMDB (including year of release)
- Genders are pipe-separated and are selected from the following genres:
  - Action
  - Adventure
  - Animation
  - Children's
  - Comedy
  - Crime
  - Documentary
  - Drama
  - Fantasy
  - Film-Noir
  - Horror
  - Musical
  - Mystery
  - Romance
  - Sci-Fi
  - Thriller
  - War
  - Western

Chapter 10. Design of the Metrics.

The evaluation metrics we want to use to evaluate the system need some data structures to get the results. In this chapter, the structures and the motivations to use them are explained.

10.1. Training dataset, Test dataset and Estimated dataset.

These components are the basic structures of the similarity evaluation metrics:

- The Training dataset stores the ratings to calculate the similarity between users and to generate the recommendations that are used in the metrics. This data is obtained from the dataset that the system is working with.
- The Test dataset stores some ratings from the system source dataset that are selected randomly. These ratings are not used in the Training dataset to help to generate recommendations. They will appear as unrated items (by the user that give the rate).
- The Estimated dataset stores the predicted recommendation values for the ratings stored in the Test dataset.

10.2. Relevant items and selected items.

We see the definition of these concepts in the analysis of evaluation metrics. As said before, when the scale is not binary it is necessary to transform the rating values in a binary scale establishing a threshold.

As shown in the chapter before, the scale of the ratings used in the test files is from 1 to 5. The MovieLens dataset is commonly transformed into a binary scale by converting every rating of 4 or 5 to "relevant" and all ratings of 1–3 to "not relevant" [DKHGBR98].

Now, we can say that there is a relevant item for each rating stored in the Test dataset which has a value equal or higher to 4. In the other hand, we can define a selected item as each predicted recommendation value that overcame the threshold to be consider as relevant.

Chapter 11. RS implementation.

In this chapter the principal characteristics of the implementation of the system are explained. One of the most important aspects of the implementation of the system is the kind of structure to store the large amounts of data (users, items, ratings and similarities).

The main criterion for the choice was to save the largest possible computation time. Because of this, the main data structures are implemented with the class of the .NET framework *Dictionary(TKey, TValue)*.

The *Dictionary (TKey, TValue)* generic class provides a mapping from a set of keys to a set of values. Each addition to the dictionary consists of a value and its associated key. Retrieving a value by using its key is very fast, close to *O(1)*, because the *Dictionary (TKey, TValue)* class is implemented as a hash table[4].

Another advantage of the class *Dictionary (TKey, TValue)* is its enumeration properties. It is possible to access to all the stored pairs of key-value with the instruction *foreach*.

11.1. Users.

For the implementation of the class *User*, the most of attributes were commented in the design.

```
class User
{
    public int id;

    // Profile information
    private string gender;
    private int age;
    private int occupation;
    private string zipCode;

    public double ratingAverage;
    public Dictionary<int,int> ratings; //<item.id,rating>
    public Dictionary<int,int> similarityEnhanced; //<user.id,value>
    public Dictionary<int,double> recommendedItems; //<item.id,value>
    public List<int> skippedItems; //<item.id,item.id,...>
    public List<int> skippedUsers; //<user.id,user.id,...>
...
```

[4] http://msdn.microsoft.com

The only one attribute added in implementation time is the *ratingAverage.* This attribute stores the average of the ratings of the user. It is possible to calculate this value in any moment from the list of ratings. As a consequence, an extra computational time is required. The value only changes when the user rates a new film. According to that, the final decision has been to store this value in an attribute and to recalculate it each time the user modifies his or her ratings.

The attribute *ratings* is implemented with a *Dictionary<int,int>* where the *Key* is the id of the item and the *Value* is the value of the rating. It gives instant access to the rating of the user for a specific item.

Here are commented the most relevant details of implementation about the methods of the class *User*.

```
public void RecalculateRatingParameters()
{
    ratingAverage = 0;
    foreach (KeyValuePair<int, int> r in ratings)
        ratingAverage += r.Value;
    ratingAverage = ratingAverage / ratings.Count;
}
```

Other interesting methods of the class user are the followings:

```
public double RatingAverageWithSkippedItems()
{
    double totalValue = 0;
    int itemValue;
    foreach (int itemId in skippedItems)
    {
        if (ratings.TryGetValue(itemId, out itemValue))
            totalValue += (ratingAverage - itemValue) /
(ratings.Count - skippedItems.Count);
    }
    return ratingAverage + totalValue;
}


public double RatingAverageWithSkippedItems(List<int> skippedItems)
{
    double totalValue = 0;
    int itemValue;
    foreach (int itemId in skippedItems)
    {
        if (ratings.TryGetValue(itemId, out itemValue))
            totalValue += (ratingAverage - itemValue) /
(ratings.Count - skippedItems.Count);
    }
    return ratingAverage + totalValue;
}
```

The method *RatingAverageWithSkippedItems is overloaded*. The method obtains the rating average using the value calculated (ratingAverage) as base. Then, it modifies this value using the values of the ratings of the skipped items.

The first method uses the list of skippedItems of the own user that executes the method. It is to calculate the own ratingAverage of the user with his or her skippedItems.

But, as said before, the similarities formulas compare two users: the active user (u) and a neighbour (n). For the calculation of the similarity, it is necessary the rating average of the user (n) who is comparing to, but excluding for the calculation the skipped values of the user (u). The second method calculates the average rating of the owner but skips the items it receives as parameter from another user.

```
public Dictionary<int, int> CommonRatings(User otherUser)
{
    Dictionary<int, int> cr = new Dictionary <int,int>();
    foreach (KeyValuePair<int, int> r in ratings)
        if (otherUser.ratings.ContainsKey(r.Key)&&
           (!skippedItems.Contains(r.Key)))
            cr.Add(r.Key, r.Value);
    return cr;
}
```

The method CommonRatings returns a list with the rating two users (the owner and another received as parameter) has in common. The method does not add the ratings of the skipped items list of the owner even if it is a common item with the other user.

11.2. Items.

The class Item mostly keeps the structure from the design with small changes.

```
class Item
{
    public int id;
    public string title;
    public Dictionary<int, int> ratings = new Dictionary<int, int>();
                                              //<user.id, rating>
    private double ratingsVariance;
...
```

The only one attribute added in implementation is the *ratingVariance.* The motivation of added is the same that the ratingAverage of the class user. This attribute stores the variance of the ratings of the item. This value is possible to calculate in any moment from the list of ratings, but it supposes an extra computational time.

The attribute *ratings* is implemented with a *Dictionary<int,int>* where the *Key* is the id of the user and the *Value* is the value of the rating. It lets instant access to the rating of the user for a specific item.

11.3. Collaborative filtering system.

This class is the most complex of the system. Its structure is the following:

```
class CFSystem
{
    public double averageVariance;
    public Dictionary<int, User> users; //<user.id, user>
    public Dictionary<int, Item> items;//<item.id, item>
    public Dictionary<int, Dictionary<int, double>> similarities;
    public Dictionary<int, Dictionary<int, double>> metricTestSet,
metricEstimatedSet;
    private Random r;
…
```

It contains a list for the users and another one for the items, both of them implemented with the type Dictionary. It has even a two-dimensional dictionary for the similarities and two more to execute the metrics designed (metricTestSet and metricEstimatedSet). The function of the last two list is explained before in the design of the metrics.

11.3.1. Similarity formulas and recommendation value.

Here is the implementation of the different similarities formulas:

```
public double UserSimilarityFormula1(User u1, User u2)
{
    int u1Value, u2Value;
    double num = 0, denom1 = 0, denom2 = 0;
    Dictionary<int, int> cr = u1.CommonRatings(u2);
    if (cr.Count == 0)
        return 0;
    double u1RatingAverage = u1.RatingAverageWithSkippedItems();
    double u2RatingAverage =
              u2.RatingAverageWithSkippedItems(u1.skippedItems);
    foreach (KeyValuePair<int, int> itemRating in cr)
    {
        u1Value = cr [itemRating.Key];
        u2Value = u2[itemRating.Key];
        num += (u1Value - u1RatingAverage) *
```

```
                (u2Value - u2RatingAverage);
        denom1 += (u1Value - u1RatingAverage) *
                  (u1Value - u1RatingAverage);
        denom2 += (u2Value - u2RatingAverage) *
                  (u2Value - u2RatingAverage);
    }
    if ((denom1 == 0) || (denom2 == 0))
        return 0;
    return num / Math.Sqrt(denom1 * denom2);
}
```

In the first formula the common ratings are obtained, then the average of the items are calculated without the skipped items and finally the 3 sums of the formula are given. At the end the method returns the quotient of one of the sums divided the square root of the other two.

The second formula has the same implementation with the difference that when the sum is calculatied, the values are multiplied by the variance.

```
public double UserSimilarityFormula2(User u1, User u2)
{
    Item item;
    int u1Value, u2Value;
    double num = 0, denom1 = 0, denom2 = 0;
    Dictionary<int, int> cr = u1.CommonRatings(u2);
    if (cr.Count == 0)
        return 0;
    double u1RatingAverage = u1.RatingAverageWithSkippedItems();
    double u2RatingAverage =
u2.RatingAverageWithSkippedItems(u1.skippedItems);
    double averageVariance =
AverageVarianceWithSkippedItems(u1.skippedItems);
    foreach (KeyValuePair<int, int> itemRating in cr)
    {
        u1Value = cr [itemRating.Key];
        u2Value = u2[itemRating.Key];
        items.TryGetValue(itemRating.Key, out item);
        num += (u1Value - u1RatingAverage) *
               (u2Value - u2RatingAverage) *
               item.RatingsVariance() / averageVariance;
        denom1 += (u1Value - u1RatingAverage) *
                  (u1Value - u1RatingAverage) *
                  item.RatingsVariance() / averageVariance;
        denom2 += (u2Value - u2RatingAverage) *
                  (u2Value - u2RatingAverage) *
                  item.RatingsVariance() / averageVariance;
    }
    if ((denom1 == 0) || (denom2 == 0))
        return 0;
    return num / Math.Sqrt(denom1 * denom2);
}
```

The user similarities formula 3 works differently from the previous one. The sum of the denominator is calculated before the calculation of the sum of the numerator, because each element of the sum of the numerator is divided by the sum of the denominator calculated.

```
public double UserSimilarityFormula3(User u1, User u2)
{
    int u1Value, u2Value;
    double num = 0, denom1 = 0, denom2 = 0;
    Dictionary<int, int> cr = u1.CommonRatings(u2);
    if (cr.Count == 0)
        return 0;
    foreach (KeyValuePair<int, int> itemRating in u1.ratings)
        if (!u1.skippedItems.Contains(itemRating.Key))
            denom1 += itemRating.Value * itemRating.Value;
    foreach (KeyValuePair<int, int> itemRating in u2.ratings)
        if (!u2.skippedItems.Contains(itemRating.Key))
            denom2 += itemRating.Value * itemRating.Value;
    if ((denom1 == 0) || (denom2 == 0))
        return 0;
    foreach (KeyValuePair<int, int> itemRating in cr)
    {
        u1Value = cr [itemRating.Key];
        u2Value = u2[itemRating.Key];
        num += u1Value * u2Value / Math.Sqrt(denom1 * denom2);
    }
    return num;
}
```

The last formula works in a similar way to the formula before. The difference is that in this case the values of the sum are multiplied by the variance of each item.

```
public double UserSimilarityFormula4(User u1, User u2)
{
    Item item;
    int u1Value, u2Value;
    double num = 0, denom1 = 0, denom2 = 0;
    Dictionary<int, int> cr = u1.CommonRatings(u2);
    if (cr.Count == 0)
        return 0;
    double averageVariance =
AverageVarianceWithSkippedItems(u1.skippedItems);
    foreach (KeyValuePair<int, int> itemRating in u1.ratings)
    {
        if (!u1.skippedItems.Contains(itemRating.Key))
        {
            items.TryGetValue(itemRating.Key, out item);
            denom1 += itemRating.Value * itemRating.Value *
item.RatingsVariance() / averageVariance;
        }
    }
    foreach (KeyValuePair<int, int> itemRating in u2.ratings)
    {
        if (!u1.skippedItems.Contains(itemRating.Key))
        {
            items.TryGetValue(itemRating.Key, out item);
            denom2 += itemRating.Value * itemRating.Value *
item.RatingsVariance() / averageVariance;
        }
    }
    if ((denom1 == 0) || (denom2 == 0))
        return 0;
    foreach (KeyValuePair<int, int> itemRating in cr)
    {
        u1Value = cr [itemRating.Key];
        u2Value = u2[itemRating.Key];
        item = items[itemRating.Key];
        num += u1Value * u2Value * item.RatingsVariance() /
               (Math.Sqrt(denom1 * denom2) * averageVariance);
    }
    return num;
}
```

Compared to the implementation of the recommendations generation, the following method visits all the items of the system. For each one that is not rated by the user, the recommendation value is calculated and is stored in the list of recommended items of the user.

```
public void CalculateRecommendations(User u)
{
    u.recommendedItems.Clear ();
    foreach (KeyValuePair<int, Item> pair in items)
        if (!u.ratings.ContainsKey(pair.Key))
            u.recommendedItems.Add(pair.Key, RecommendationValue(u,
pair.Value));
}
```

PART IV

RESULTS AND CONCLUSIONS

Chapter 12. Test: time calculations.

The recommender systems are characterized by managing large dataset. One of the most important challenges for them after giving good recommendations, is to work with this amount of data in a reasonable computing time.

In order to test how the built system works from the computational time point of view, we run some tests as explained in the previous sections.

Some datasets with the following characteristics were run:

	FILE 1	FILE 2	FILE 3	FILE 4	FILE 5	FILE 6	FILE 7	FILE 8
Users	100	500	1000	2000	3000	4000	5000	6000
Items	100	500	1000	1500	2000	2500	3000	3900
Ratings	457	10186	37604	131478	263516	405294	477727	637776
Ratings / user	4,57	20,37	37,60	65,73	87,83	101,32	95,54	106,29

The similarities between the users and the recommended ratings were calculated with each dataset for each user. Then the calculated similarities were saved in a file and loaded again to calculate the time to read this information from it.

It is useful because the system does not have to calculate the similarities all the times they are executed. Reading information from a file is faster than calculating the similarities.

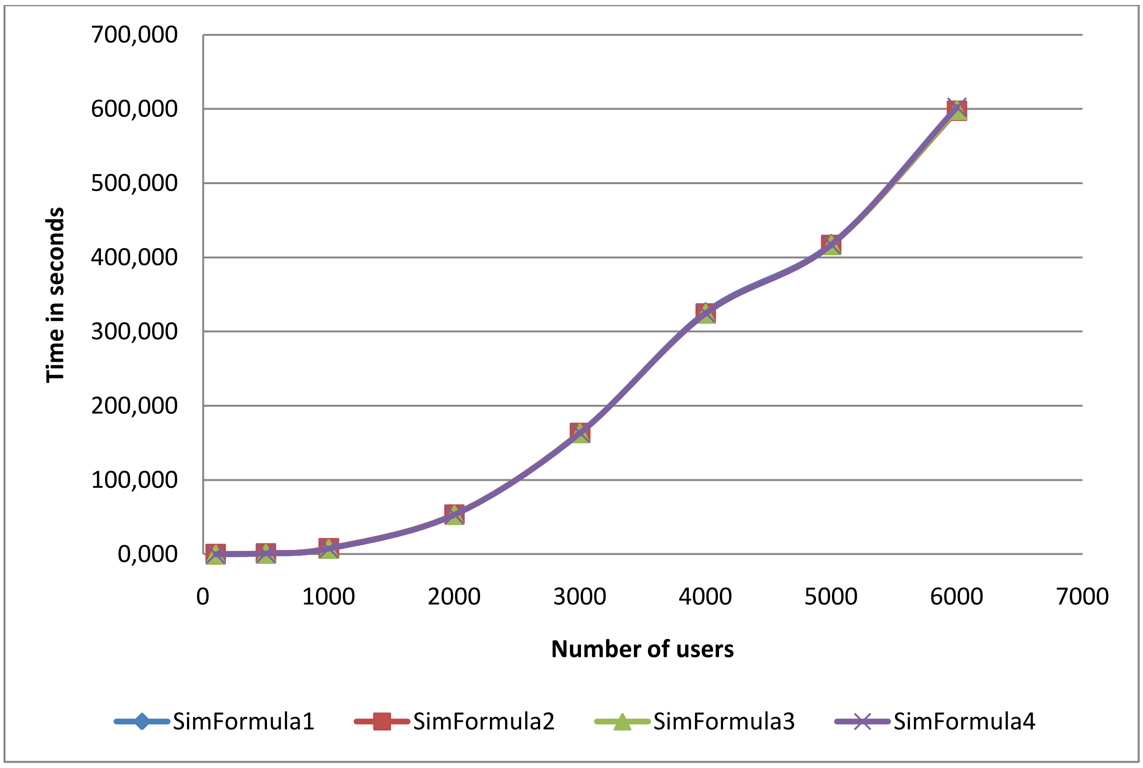
12.1. Time to calculate the similarities.

The time to calculate the similarities for each file appears in the following table:

	FILE 1	FILE 2	FILE 3	FILE 4	FILE 5	FILE 6	FILE 7	FILE 8
SimFormula1	0,024	1,116	7,60	53,2	163,1	325,7	417,6	597,7
SimFormula2	0,013	1,112	7,69	53,3	163,3	324,4	416,8	597,4
SimFormula3	0,020	1,107	7,59	53,3	163,5	324,6	417,0	598,8
SimFormula4	0,013	1,125	7,58	53,2	163,4	324,7	417,1	602,3

As it can be supposed, the time to compute the similarities will be O(n^2) if we consider n as the number of users, because each user has to be compared with the rest of them. The final number of similarities will be *numberUsers^2.* The half of the similarities can be considered only in other recommender systems where they are symmetric. In this case rather, as explained before, the feedback techniques can give asymmetric similarities between two users.

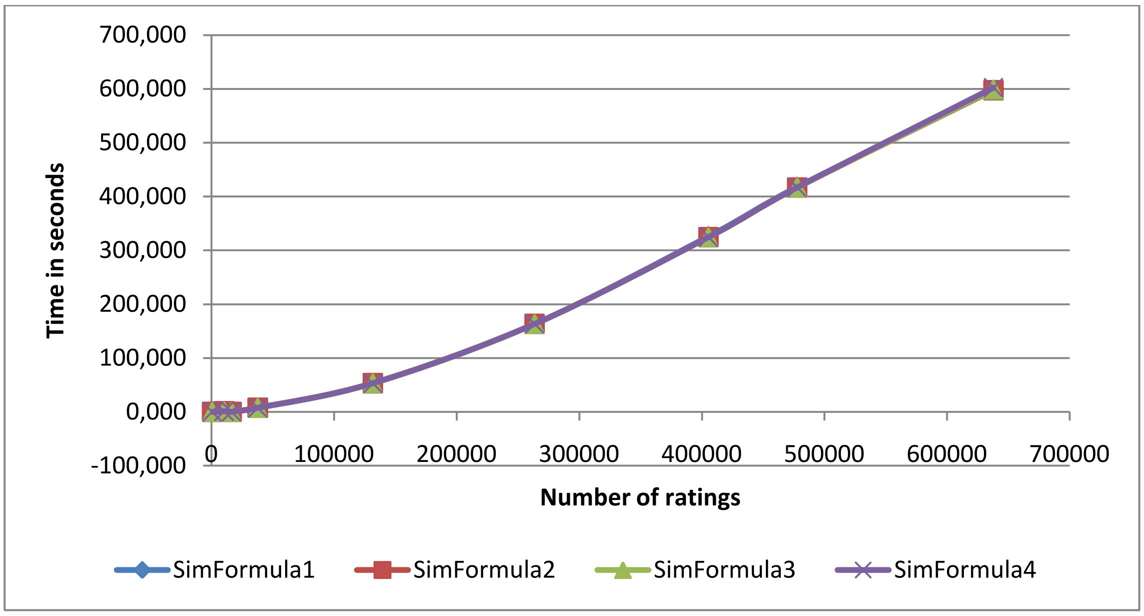
Comparing the similarities formulas, the difference between them in the calculations is minimum. The following graph shows how the four formulas are joined in one.



Graph4. Time to calculate the similarity against the number of users in the database.

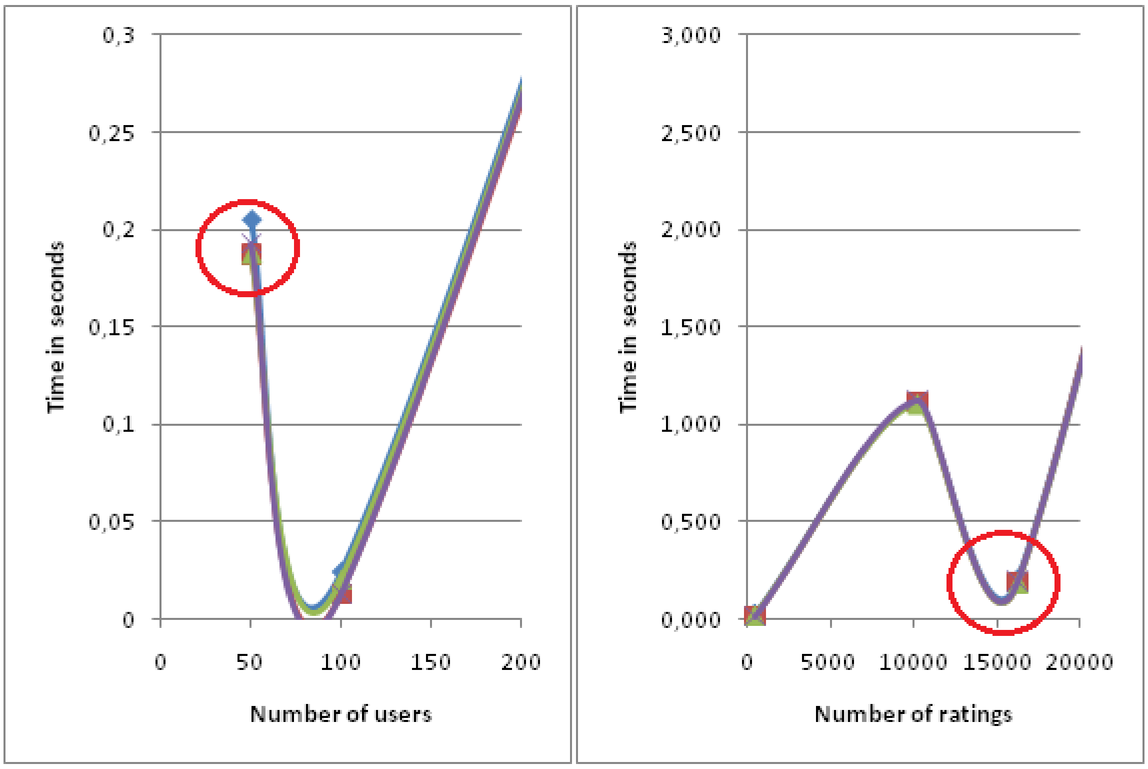
In the graph it is possible to see how "strange" the behaviour of the function in the file 7 is. If we check the **¡Error! No se encuentra el origen de la referencia.** in file 7, we can find out what is "strange": the *rating for each user* does not follow the ascendant grow as in the previous files.

It can mean that the time of computing the similarities is also related with the total number of votes that the users rates. In the next graph we can see the relation between the time and the number of ratings. It is a mostly linear relation.



Graph5. Time to calculate the similarity against the number of ratings in the database.

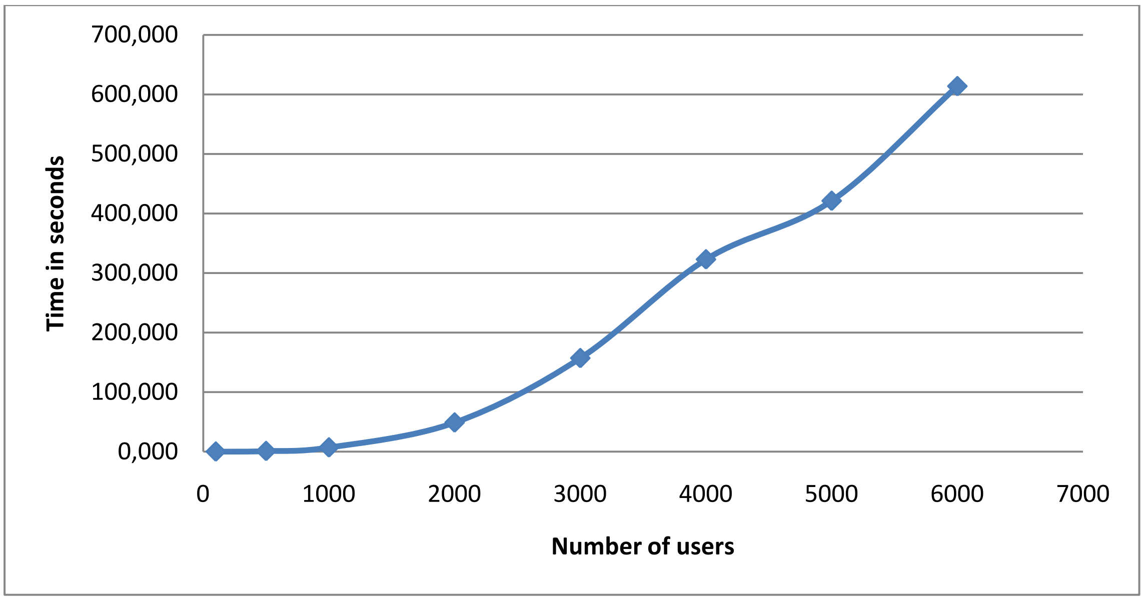
In the previous case, all the files are more or less proportional in number of users, films, ratings. We add a strange case that breaks absolutely with the previous proportion of user/rating. We run a case with 50 users and 3952 users. It supposes 16207 ratings. The time to compute each similarity formula is: 0,205 - 0,187 - 0,187 - 0,193. If we zoom in the part of the graph of the new file, we can see how it generates a "strange" behaviour in the function.



Graph6.

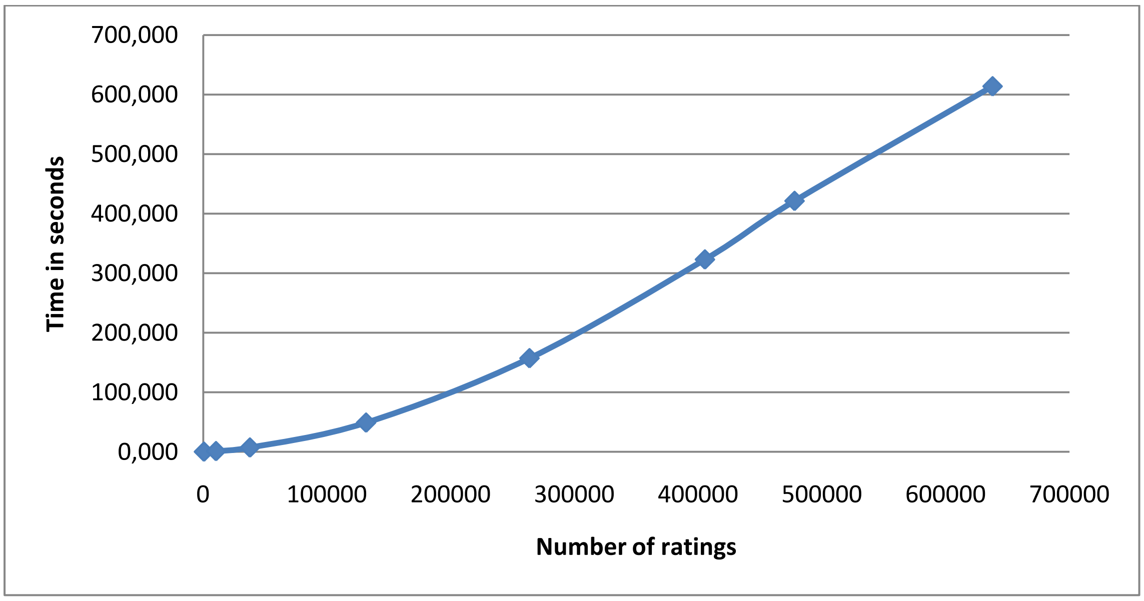
The “strange” behaviour of the function with both variables (number of users and number of ratings) means that the time does not depend only on one of them but both are significant.

A similar situation appears to calculate the recommendations. For each recommendation for each user, it is necessary to read the similarities of the user with the rest of users, which supposes a computational time O(n^2) where n is number of users.



Graph7. Time to calculate the recommendations for users against the number of users in the database.

But we can see in the following graph that the relation between the time to calculate the recommendations and the number of ratings is mostly linear.



Graph8. Time to calculate the recommendations for users against the number of ratings in the database.

Chapter 13. Test: Similarities.

In the test over the similarities formulas the designed evaluation metrics were run 100 times for each file. In order to determine whether there are differences between the different formulas, we calculate the average value and the standard deviation of each metric for each file. These values are used to hypothesis contrast over the difference of averages of all the pairs of similarities formulas.

The values calculated by the system were studied with different datasets that can are created with the original data of the MovieLens files.

13.1. Different sizes of dataset.

This group of datasets contains the original data from the MovieLens files. To generate each one of the files *n_users_ m_movies* , we selected the first *n* users of the files, the first *m* items and the rating of these users for the selected items.

The aim of this test is to study how the quality of the recommendation evolves when the number of users, items and ratings increase.

Then, the different evaluation metrics were run with each one of the files, obtaining the following results:

13.1.1. Deviation.

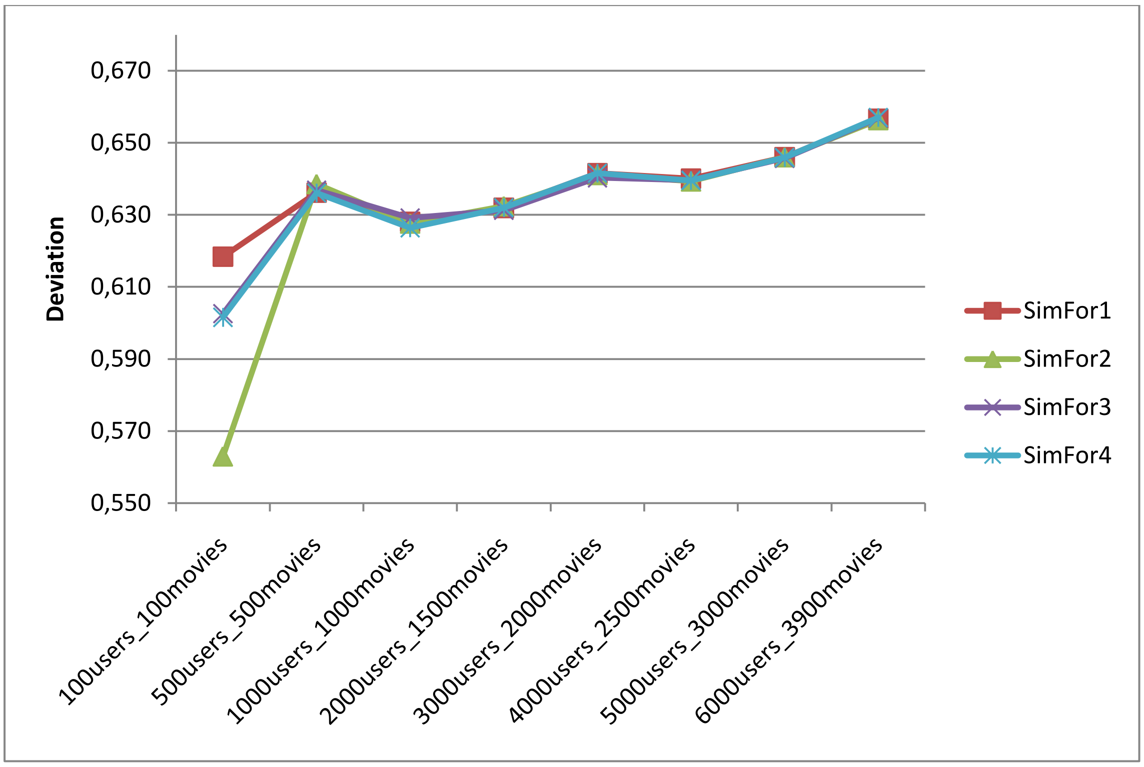
For the Deviation the results values appear in the table.

File	100u 100m	500u 500m	1000u 1000m	2000u 1500m	3000u 2000m	4000u 2500m	5000u 3000m	6000u 3900m
SimFor1	0,618	0,636	0,628	0,632	0,642	0,640	0,646	0,657
SimFor2	0,563	0,639	0,628	0,632	0,641	0,639	0,646	0,656
SimFor3	0,603	0,637	0,629	0,631	0,640	0,640	0,646	0,657
SimFor4	0,602	0,636	0,626	0,632	0,642	0,640	0,646	0,657

Over this result, we do a hypothesis contrast means difference for each pair of values. The absolutes values higher of 1.96 indicates that the differences between formulas are relevant.

File	100u 100m	500u 500m	1000u 1000m	2000u 1500m	3000u 2000m	4000u 2500m	5000u 3000m	6000u 3900m
SimFor1-SimForm2	**3,07**	-0,79	0,28	-0,62	0,75	1,65	-0,09	0,81
SimFor1-SimForm3	0,85	-0,21	-0,74	0,77	**2,07**	0,84	0,66	-0,47
SimFor1-SimForm4	0,98	0,02	1,11	-0,07	-0,09	0,99	0,10	-0,98
SimFor2-SimForm3	**-2,06**	0,52	-1,02	1,41	1,28	-0,78	0,72	-1,31
SimFor2-SimForm4	**-2,14**	0,75	0,81	0,54	-0,83	-0,53	0,19	-1,91
SimFor3-SimForm4	0,06	0,22	1,85	-0,82	-2,15	0,20	-0,58	-0,50

As it is shown in the table, the similarity formula 2 give recommendation with significant less deviation than the others. In the graphic representation we can see the difference between formulas in the first file.



Graph9. Deviation for each formula

13.1.2. Precision.

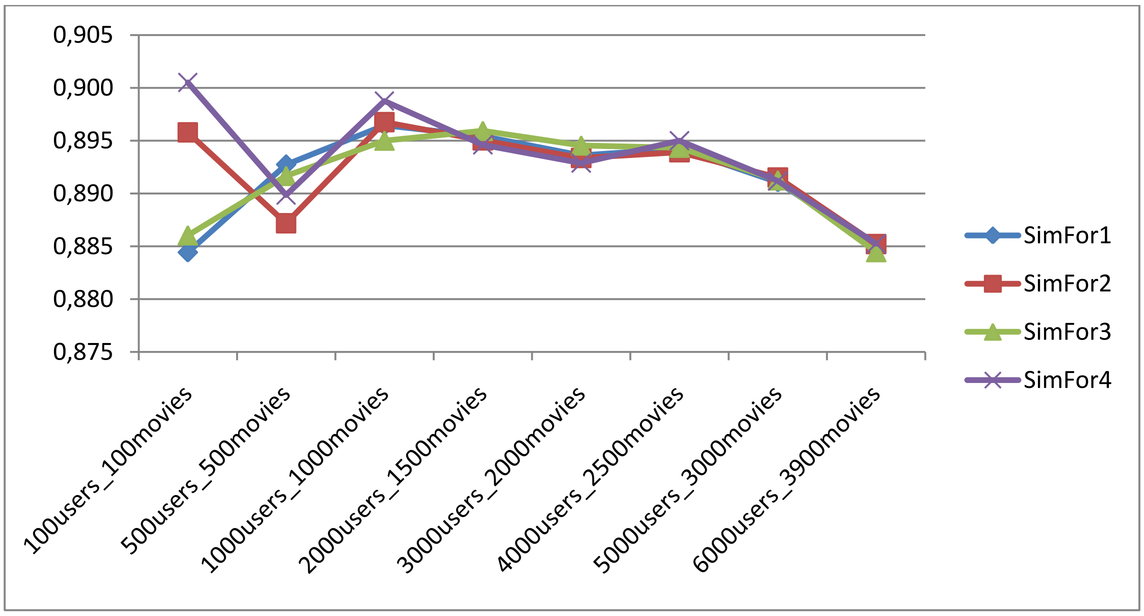
The results obtained are:

File	100u 100m	500u 500m	1000u 1000m	2000u 1500m	3000u 2000m	4000u 2500m	5000u 3000m	6000u 3900m
SimFor1	0,884	0,893	0,896	0,895	0,894	0,894	0,891	0,885
SimFor2	0,896	0,887	0,897	0,895	0,893	0,894	0,891	0,885
SimFor3	0,886	0,892	0,895	0,896	0,895	0,894	0,891	0,884
SimFor4	0,901	0,890	0,899	0,895	0,893	0,895	0,891	0,885

After these results we calculate the values for the hypothesis contrast:

File	100u 100m	500u 500m	1000u 1000m	2000u 1500m	3000u 2000m	4000u 2500m	5000u 3000m	6000u 3900m
SimFor1-SimForm2	-0,66	1,69	-0,24	0,46	0,41	0,73	-0,97	-0,48
SimFor1-SimForm3	-0,09	0,35	1,13	-0,64	-1,73	-0,16	-0,39	1,48
SimFor1-SimForm4	-0,95	0,90	-1,67	1,04	1,29	-1,45	-0,23	-0,57
SimFor2-SimForm3	0,56	-1,48	1,31	-1,08	**-2,04**	-0,95	0,55	**2,17**
SimFor2-SimForm4	-0,29	-0,84	-1,39	0,54	0,83	**-2,30**	0,70	-0,09
SimFor3-SimForm4	-0,85	0,62	**-2,59**	1,69	**2,95**	-1,36	0,15	**-2,25**

As we can see, there are many significant values, but it is very difficult to get a conclusion from them. There are differences between the similarity formulas 2, 3 and 4 but in some cases the precision is significant better in one formula and with other files it is the opposite.



Graph10. Precision of each formula

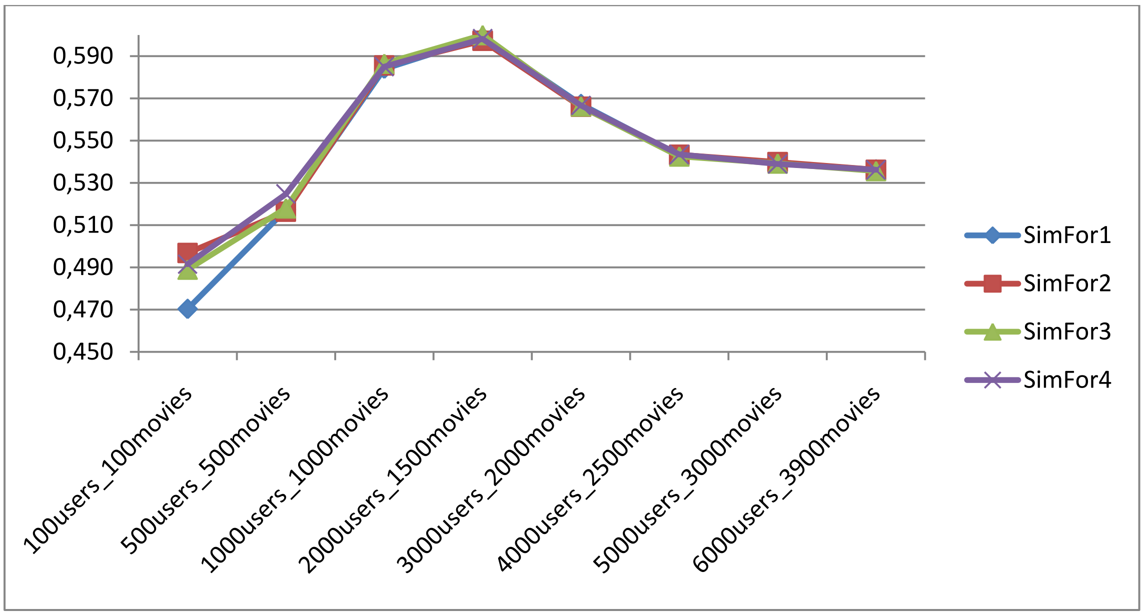
13.1.3. Recall.

The results obtained for the recall metric are:

File	100u 100m	500u 500m	1000u 1000m	2000u 1500m	3000u 2000m	4000u 2500m	5000u 3000m	6000u 3900m
SimFor1	0,470	0,517	0,584	0,599	0,567	0,543	0,539	0,536
SimFor2	0,497	0,516	0,586	0,597	0,566	0,543	0,540	0,536
SimFor3	0,489	0,518	0,586	0,600	0,566	0,543	0,539	0,536
SimFor4	0,491	0,525	0,585	0,598	0,567	0,543	0,539	0,536

After these results we calculate the values for the hypothesis contrast:

File	100u 100m	500u 500m	1000u 1000m	2000u 1500m	3000u 2000m	4000u 2500m	5000u 3000m	6000u 3900m
SimFor1-SimForm2	-1,30	0,22	-0,80	1,55	1,57	-0,43	-0,87	0,28
SimFor1-SimForm3	-0,92	-0,17	-1,23	-1,16	1,22	0,69	0,21	1,34
SimFor1-SimForm4	-1,00	-1,82	-0,53	0,57	0,81	-0,64	0,64	0,27
SimFor2-SimForm3	0,43	-0,40	-0,44	**-2,66**	-0,28	1,16	1,09	0,97
SimFor2-SimForm4	0,28	**-2,02**	0,26	-0,92	-0,84	-0,23	1,48	-0,01
SimFor3-SimForm4	-0,13	-1,74	0,70	1,66	-0,50	-1,35	0,45	-0,97



Graph11. Recall of each formula

13.1.4. F1 metric.

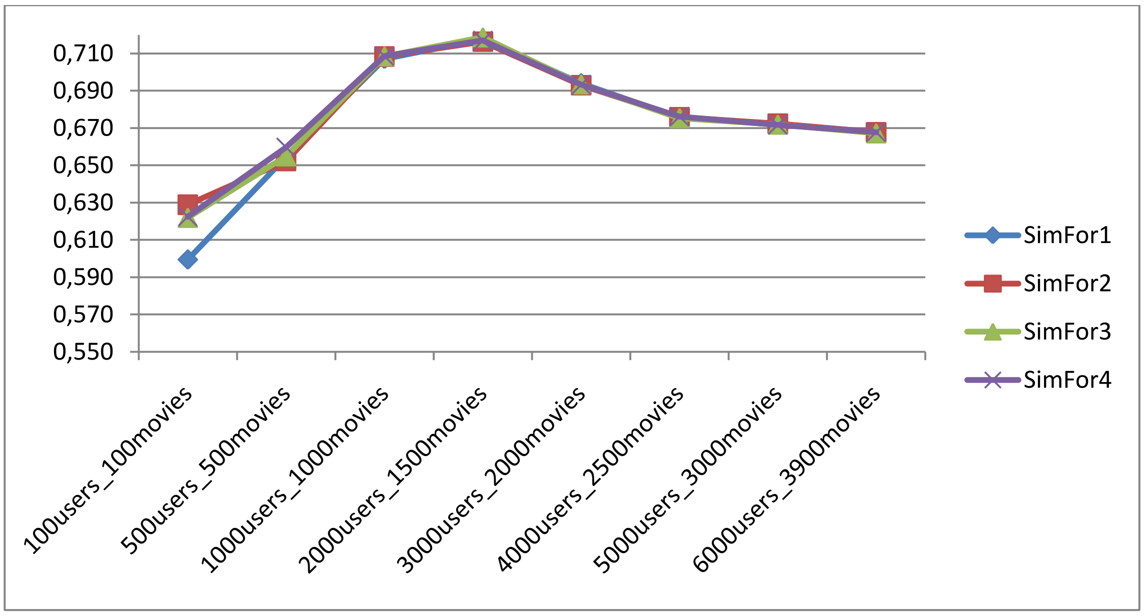
The results obtained are:

File	100u 100m	500u 500m	1000u 1000m	2000u 1500m	3000u 2000m	4000u 2500m	5000u 3000m	6000u 3900m
SimFor1	0,599	0,655	0,707	0,718	0,694	0,676	0,672	0,668
SimFor2	0,629	0,652	0,708	0,716	0,693	0,676	0,672	0,668
SimFor3	0,622	0,655	0,708	0,719	0,693	0,675	0,672	0,667
SimFor4	0,623	0,660	0,709	0,717	0,693	0,676	0,672	0,668

After these results we calculate the value for the hypothesis contrast:

File	100u 100m	500u 500m	1000u 1000m	2000u 1500m	3000u 2000m	4000u 2500m	5000u 3000m	6000u 3900m
SimFor1-SimForm2	-1,56	0,62	-0,79	1,62	1,58	-0,22	-1,07	0,14
SimFor1-SimForm3	-1,18	-0,10	-0,84	-1,25	0,72	0,59	0,09	1,66
SimFor1-SimForm4	-1,20	-1,48	-0,93	0,88	1,10	-0,99	0,54	0,11
SimFor2-SimForm3	0,41	-0,75	-0,07	**-2,80**	-0,86	0,84	1,23	1,45
SimFor2-SimForm4	0,37	**-2,08**	-0,15	-0,71	-0,51	-0,83	1,62	-0,03
SimFor3-SimForm4	-0,04	-1,47	-0,08	**2,08**	0,37	-1,61	0,48	-1,50

There are some significant differences but without any pattern that lets say that one similarity formula is better than other one. Here we can see the graphical representation:



Graph12. F1 for each formula

Chapter 14. Conclusions.

After the analysis of the entire test running in the system with different datasets, we can get some conclusions that are describe in the following paragraphs.

- The time to compute the similarities between all the users is O (n^2) for the numbers of users and mostly lineal for the number of ratings that are in the system. The solution acquired is to store the calculated similarities in files, that lets recover the information later. Future improvements of the application could work in this way, exploring some techniques like clustering or some alternatives to don't have to recalculate all the similarities when a user rates a new item.

- The different similarities formulas get very similar results with enough number of ratings. The differences between use one formula or other is minimum. Only in specific scenarios with few ratings, they can give differents result.

- One alternative to influence in the results of the recommendations is to use feedback from the user. Deleting items or items from the calculation or enhancing in a positive or negative way the similarity with others users, the user can modify

Bibliography.

- ***[A09]*** Ira Assent "Actively building private recommender networks for evolving reliable relationships", in Proc. Workshop Modeling, Managin and Mining of Evolving Social Networks (M3ESN) at IEEE ICDE, 2009.

- ***[AT05]*** G. Adomavicius and A. Tuzhilin "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions" IEEE transactions on knowledge and data engineering, vol. 17, no. 6, 734-749, 2005.

- ***[BE97]*** Danah M. Boyd, Nicole B. Ellison "Social Networks Sites: Definition, History and Scholarship", Michigan State University, 2007.

- ***[BHK98]*** Breese, J.S., Heckerman, D., Kadie, C.: Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In Proceeding of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI). (1998) Madison, Wisconsin. Morgan Kaufmann p. 43-52.

- ***[BM05]*** M. Bilgic and R. J. Mooney. Explaining recommendations: Satisfaction vs. promotion. In Beyond PersonalizationWorkshop, IUI, 2005.

- ***[BS97]*** M. Balabanovic and Y. Shoham, "Fab: Content-Based, Collaborative Recommendation", Comm. ACM, vol. 40 no. 3, pp. 66-72, 1997.

- ***[CR06]*** A. Ceglar and J. Roddick, "Association mining," ACM Computing Surveys (CSUR), vol. 38, no. 2, p. 5, 2006.

- ***[DKHGBR98]*** DAHLEN, B. J.,KONSTAN, J. A., HERLOCKER, J. L., GOOD, N., BORCHERS, A., AND RIEDL, J. 1998. Jumpstarting movielens: User benefits of starting a collaborative filtering system with "dead data". University of Minnesota.

- ***[EKSWX98]*** Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, Xiaowei Xu "Incremental Clustering for Mining in a Data Warehousing Environment"

- ***[HKTR04]*** J.L. Herlocker, J.A. Konstan, L.G. Terveen, and J.T. Riedl, "Evaluating Collaborative Filtering Recommender Systems," ACM Trans. Information Systems, vol. 22, no. 1, pp. 5-53, 2004.

- ***[KNNDK08]*** V. Krishnan, P. K. Narayanashetty, M. Nathan, R. T. Davies, and J. A. Konstan, "Who predicts better?: Results from an online study comparing humans and an online recommender system" in Proceedings of the 2008 ACM Conference on Recommender Systems, 2008.

- ***[SFHS97]*** Schafer, B., Frankowski, D., Herlocker, J., Sen, S.: Collaborative Filtering Recommender Systems. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.): The Adaptive Web: Methods and Strategies of Web Personalization. Lecture Notes in Computer Science, Vol. 4321. Springer-Verlag, Berlin Heidelberg New York (2007)

- ***[SK09]*** Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. Advances in Artificial Intelligence, 2009:1–20, 2009

- ***[RISBR94]*** Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Rield, J. (1994). "GroupLens: An open architecture for collaborative filtering of netnews". In Proceedings of the ACM 1994 Conference of Computer Supported Cooperative Work, pages 175-186, New York, ACM.