Università
Ca’Foscari
Venezia

Master’s Degree

in Computer Science

Final Thesis

# Enhancing a Hybrid Collaborative Filtering Recommender by Exploiting Approximate Top-k Binary Patterns

**Supervisor**
Prof. Salvatore Orlando

**Graduand**
Andrea Furlan
877704

**Academic Year**
2019 / 2020

# Acknowledgments

I would like to acknowledge you Erica, my girlfriend. Not only for having always been by my side during these academic years but also for having pushed me to always give my best, always putting up with me despite everything. You has always been by my side, stimulating my research, soothed my anger, reducing my worries, reading and correcting every line of this thesis with patience. Thank you infinitely and eternally, my Love.

I would like to thank my family for helping me endure the dark times of my academic career. Without you, I wouldn't be where I am today, you are the foundation of my life. Thanks mom, dad and Stefano.

I would like to thank my supervisor Professor Salvatore Orlando, not only for having been the guide in this thesis, but also for having always remained a human and sensitive person towards me.

I would like to thank all my friends who gave me a call to hear my reasoning or for a simple chat.

The last thanks goes to you grandmother, you have always believed in me and you have always told me ”sei una cima”, you have been my strength in these last months, thank you.

# Contents

# Chapter 1

# Introduction

The amount of content available on the web today is enormous. Just think that in sixty seconds on the web in the 2019, 1 million Facebook log-in happened, or again 1 million $ have been spent in Amazon or 3,8 million search queries have been done. If we think how much data has been created only in 1 minute, it's impressive [15]. This is far too much content for any person to consume, and it is called the problem of "information overload". To help people to decide what items might be worthwhile to look at or not, people need some tools to help them. One effective tool for this task is a recommender system. These systems suggest items that a user might be interested based on his preferences, observed behaviors, and information about the items themselves.

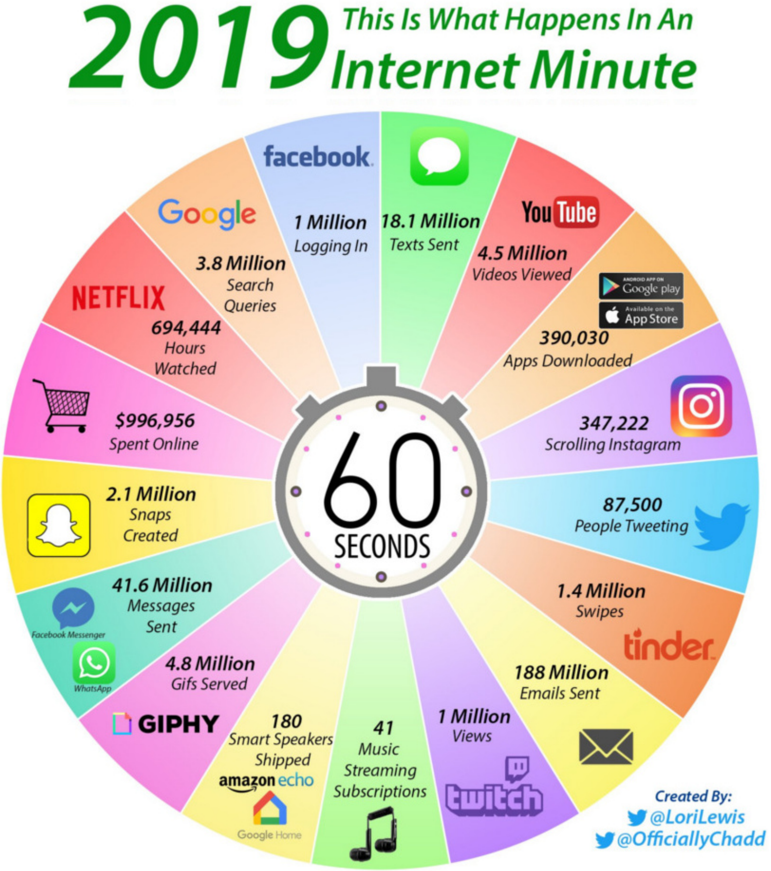


Figure 1.1: What happens in 60 seconds? [15]

Collaborative Filtering (CF)-based recommender systems are indispensable tools to find items of interest from a great number of available items. Moreover, companies that deploy a CF-based recommender system may be able to increase own profits by drawing customers' attention to items that they are likely to buy.
However, the great number of users and items typical in e-commerce systems demand specially designed recommender algorithms that can gracefully cope with the vast size of the data. Many algorithms have been proposed thus far, where the principal concern are recommendation quality, but they may be too expensive to operate in large-scale systems.

In this thesis we studied, analyzed and modified *ClustKnn: a Highly Scalable Hybrid Model- & Memory-Based CF Algorithm* proposed by Karypis et Al [11, 12], a simple and intuitive algorithm. This algorithm is well suited for large datasets.

The method first compresses data tremendously by building a straightforward but efficient clustering model then it generates quickly recommendations using a simple Nearest Neighbor-based approach. However we found different ways to improve this algorithm and we found some leaks in the publication and in their results. Anyhow, as Karypis et Al said, they demonstrated the feasibility of ClustKnn both analytically and empirically, and they also compared it with a number of other popular CF algorithms that, apart from being highly scalable and intuitive, it provides very good recommendation accuracy as well.

We improved over ClustKnn by adding a further step to the clustering phase. We used the $PaNDa^{+}$ framework [8] which is able to discover high quality patterns from binary datasets.

# Chapter 2

# Recommender System

Over the past few decades, with the rapid rise of Youtube, Amazon, Netflix and many other similar web services, the amount of data on the web is become enormous. Just think that, only in 60 seconds, from a research in 2019, users reached 4.19 millions search queries on Google, 694.444 hours on Netflix, 4.5 million videos on Youtube viewed and so on.

From e-commerces, to online advertising, or again to video sharing or video streaming platforms: the number of possible choices has become enormous. For example, to make the right choice like a specific movie to watch or a specific item that you are interested to buy, is very difficult to find out.
To deal with this problem, a sort of information filtering must be used in order to manipulate the enormous quantity of raw data, reduce the noise, and find out relevant data: the solution is the so-called Recommendation Systems.

A recommender system is a kind of information filtering system that tries to predict the "preference" or "rating" a user would give to a specific item. Recommender Systems (RS) are thus software tools and techniques that provide suggestions to a user.
So, recommendations or suggestions, typically, make it easier for users to access content they are interested in, and surprise/show them with items they would have never searched for, most of the times to new offers in the field of e-commerces.

The study of recommender systems is relatively new compared to research into other classical information system tools and techniques like databases or search engines.
Recommender systems emerged as an independent research area in the middle of 1990s, and in recent years, the interest in recommender systems has dramatically increased, as the following facts dimostrate:

1. Recommender systems are very important for Internet sites like Amazon.com, YouTube, Netflix, Tripadvisor, Spotify, IMDb and many others. Moreover, many media companies are still developing, deploying, or just improving RSs, as part of the services they provide. An example is the "The Netflix Prize": Netflix, the famous streaming media service, started in 2007 an open com-

petition for the best RS collaborative filtering algorithm, awarding a million dollar prize to the team that first succeeded in highly improving the performance of its recommender system. The competition was cancelled in 2010 due to a lawsuit.

2. There are many conferences and workshops dedicated to this specific field. In particular, there is a famous conference called ACM. As they said on their website, ”The ACM Conference on Recommender Systems (RecSys) is the premier international forum for the presentation of new research results, systems and techniques in the broad field of recommender systems. RecSys brings together the major international research groups working on recommender systems, along with many of the world’s leading companies active in e-commerce and other adjacent domains. It has become the most important annual conference for the presentation and discussion of recommender systems research. [...][1]”.

   Furthermore, frequently there are session dedicated to RSs during the more traditional conferences in the area of data bases, information systems and adaptive systems.

   Among these conferences, we can mention Special Interest Group on Information Retrieval (SIGIR), User Modeling Adaptation and Personalization (UMAP), ACM’s Special Interest Group on Management Of Data (SIGMOD), Knowledge Discovery and Data Mining (KDD) and World Wide Web conference (WWW).

3. In the institutions of higher education around the world, exists undergraduate and graduate courses dedicated entirely to RSs; tutorials on RSs are very popular at computer science conferences; there are a lot of tutorials online to learn the first rules to build a RS for building your own RS or real code; and a lot of books about RSs techniques were published.

Although in past few years lot of approaches to recommender systems have been developed, the interest in this field is high due to growing of practical applications in which can deal with personalized recommendation and deal with large amount of data[18].

A personalized RS technology can be used to either predict whether a particular user will like a particular product or item (prediction problem) or to identify a set of $N$ items that will be of interest to a specific user (top-N recommendation problem).

Further, many other applications use recommendation systems for different purposes like: make more profit in industry, make effective and efficient personalized result in its own system.
Nowadays companies have large amount of data available, not only user’s preferences, but the number of log-in and log-out, the time of log-in, number of clicks, etc. So in order to take advantages of this raw data, they use Intelligent Recommendation System. Many of the company used this type of system like Amazon.com, ebay.com, movies by movie lens, to make more accurate prediction of user’s mind.

There are three categories in which Recommendation Systems can be divided to create suggestions:

- *collaborative approach*: a Recommendation Systems with collaborative filtering (CF) approaches build a model of a user's past behavior (previously extracted or selected items and/or numerical ratings given to those specific items) as well as similar decisions made by other users. This model is used to predict articles or article ratings that the user might be interested in.
- *content-based approach*: a Recommendation Systems with content-based (CB) approaches are based on a description of the item and a profile of the user's preferences. These methods are best suited to situations where there is known data on an item called features (name, location, description, etc.), but not on the user. Content-based recommenders treat recommendation as a user-specific classification problem and learn a classifier for the user's likes and dislikes based on an item's features.
- *hybrid approach*: this method is a combination of the previous approaches.[13, 9]

## 2.1 Collaborative-Filtering methods

The first kind of Recommender System is the collaborative filtering (CF). The CF approach is widely used nowadays and it is based on the main idea that users who agreed in the past will agree in the future, and then, they will like similar kinds of items as they liked in the past.
In other words, if a user *"A"* has the same rating/taste/opinion as a user *"B"* on an item, then *"A"* is more likely to have same ratings/tastes/opinions of *"B"* on the other elements as well, than that of a randomly users.

So, collaborative filtering is the method of making automatic predictions (filtering) about the interests of a specific user by collecting preferences or information from many others users (collaborating).

It is important to emphasize that this method is based only on the past interactions recorded between users and items in order to produce new recommendations. In a more general sense, collaborative filtering is the process of filtering for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc. Applications that use collaborative filtering methods typically involve very large datasets.

Collaborative filtering methods have been applied to many different kinds of data, and not only in e-commerces and web applications services where the focus is on user data but also in: sensing and monitoring data, such as in mineral exploration, environmental sensing over large areas or multiple sensors; financial data, such as financial service institutions that integrate many financial sources etc.

Collaborative Filtering techniques are divided in two categories: model based and memory based approaches.

### 2.1.1 Memory based

Memory based collaborative approaches directly work with values of past interactions, these methods do not assume any latent model, and are essentially based on nearest neighbours search.
These algorithms use users rating data to compute the similarity between users or items: a typical example of this approach are neighborhood-based CF methods and it is typically divided into two classes: item-based and user-based top-N recommendations.

The neighbourhood-based algorithm calculates the similarity between two users or items, and it produces a prediction for the user by taking the weighted average of all the ratings. The calculation of similarity between items or users is very important in this approach and so it is the choice of the appropriate correlation measure.

There are multiple mathematical measures, that are used to determine how similar is a vector to a given vector: The similarity metrics mostly used are:

- Pearson similarity, a coefficient given by:

$$\text{sim(x,y)} = \frac{\sum_{i=1}^{n}(x_i-\overline{x})(y_i-\overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\overline{x})^2(y_i-\overline{y})^2}}$$

- Vector cosine based similarity, the Cosine angle between the vectors:

$$\text{sim(x,y)} = \frac{\sum_{i=1}^{n}(x_i y_i)}{\sqrt{\sum_{i=1}^{n}(x_i^2)}\sqrt{\sum_{i=1}^{n}(y_i^2)}}$$

- Euclidian Distance, the element-wise squared distance between two vectors:

$$d(x,y) = \sqrt{\sum_{i}^{n}(x_i - y_i)^2}$$

So, the principal aspects of User-Based Collaborative Filtering and Item-Based Collaborative Filtering (also called user-user and item-item) approaches are that, they use only information from the user-item interaction matrix and they assume no model to produce new recommendations. One of the biggest flaw of memory-based CF is that, these types of algorithms are not easily scalable: computing a new suggestion can be extremely time consuming for big datasets, due to the great numbers of distances to compute.

We can predict rating of user "u" for item "i" by taking weighted the sum of the ratings of item "i" from all other users (u's), where weighting is the similarity number between each user and user "u". Clearly, the similarity metric can be one of the above:

$$\widehat{r}_{u,i} = \sum_{u' \epsilon U} sim(u,u') r_{u',i}$$

whereh $u \neq u'$ and U is the KNN set.

### User-Based Collaborative Filtering

In order to make a new recommendation to a specific user, User-Based Collaborative Filtering (UB-CF) method tries to identify users with the most similar "interactions profile" (nearest neighbours) in order to suggest "new" items that are the most popular among these neighbours.
This method is called "user-user" or "user-based method" because it represents users based on their interactions with items and evaluates distances between users.

So, in order to compute a recommendation for a particular user it is necessary to:

- Find the K-nearest neighbors (KNN) to the user "u", using a similarity function "w" to measure the distance between each pair of users:

$$\textit{Similarity(u,i)} = w(u,i), i \epsilon K$$

- Predict the rating that user "u" will give to all items the k neighbors have given but "u" has not. After that, it is necessary to search for the item "j" with the best predicted rating.

Once similarities to every users have been computed, it is possible to keep the k-nearest-neighbours to the specific user and then suggest the most popular items among them.
Now, if one user "A" behaves like other users B, and C then for a product "x", A's rating is given by:

$$R_{xu} = \frac{\sum_{i=0}^{n} R_i}{n}$$

where $R_{xu}$ is the rating predicted to "x" by user "u", $i = 0$ to "n" are the users who have a behaviour similar to "u" and $R_i$ is the average of all ratings of user "i". All the "n" users are not an equal amount similar to the specific user "u", so, we found a weighted sum to provide a rank:

$$R_{xu} = \frac{\sum_{i=0}^{n} R_i W_i}{\sum_{i=0}^{n} W_i}$$

where the weights are the similarity metrics used.

### Item-Based Collaborative Filtering approach

In the case of Item-Based Collaborative Filtering approach (IB-CF), the main idea to make a new recommendation to a user, is to find items similar to the ones the user already interacted with.
A user might prefers on the basis of information collected from various other users having similar tastes or preferences. It takes into consideration the basic fact that if person X and person Y have a certain reaction for some items, then they might have the same opinion for other items too.

So, two items are considered to be similar if most of the users that have interacted with both of them did it in a similar way. This method is called "item-item" or "item-based method", because it represents items based on interactions users had with them and evaluate distances between those items.

Again, as in [17] is explained, once the set of most similar items based on the similarity measures, the next phase is to inspect the target user ratings and use a technique to obtain suggestions.

There are two main techniques that are explained[17]:

- Weighted Sum: this method computes the prediction on an item $i$ for a user $u$ by computing the sum of the ratings given by the user on the items similar to $i$. Each rating can be weighted by its corresponding similarity $s_{i,j}$.

$$P_{u,i} = \frac{\sum_{all\_similar\_items,N}(s_{i,N} * R_{u,N})}{\sum_{all\_similar\_items,N}(|s_{i,N})|}$$

  where $N$ corresponds to the set of items similar to target item and from those similar items, are picked those items which the active user has rated the item "i". The weighted sum approach is scaled by the sum of the similarity terms, in order to make sure the prediction is in the predefined range.

- Regression: This method is similar to the weighted sum approach but instead of using the rates of similar items, it uses an approximation of the ratings based on regression model.
  The basic idea is to use the same formula as the weighted sum technique, but instead of using the similar item N's "raw" ratings values $R_{u,N}$ 's, this model uses their approximated values $R'_{u,N}$ based on a linear regression model. If we indicate the respective vectors of the target item $i$ and the similar item $N$ by $R_i$ and $R_N$ the linear regression model can be expressed as:

$$\overline{R}'_N = \alpha \overline{R}_i + \beta + \varepsilon$$

  The parameters of the regression model $\alpha$ and $\beta$ are determined by going over both of the rating vectors, while $\varepsilon$ is the error of the regression model.

The main difference between User-Based Collaborative Filtering and this method is that, in this case, we directly pre-compute the similarity between the co-rated items, avoiding the K-neighborhood search.

### 2.1.2 Model based

Model based collaborative filtering approaches only depend on user-item interactions information, in order to create a model that can explain and describe these interactions.
This model is usually developed using different data mining methods and machine learning algorithms in order to predict, for example, users' rating of unrated items.
There are three main model-based CF algorithms: Non-parametric approach, Deep Learning methods, but the most popular are Matrix Factorization based algorithms.

- Non-parametric approach: among Non-parametric approaches, usually, are used simple clustering algorithms like K-Nearest Neighbours to find the K closest neighbours given a user or an item based on the similarity metrics used.
  The idea is similar as that of memory-based recommendation systems. In memory-based algorithms are used the similarities between users and/or items and use them as weights to predict a rating for a user and an item. The difference is that the similarities in this approach are computed based on an unsupervised learning model, rather than a simple similarity metric like Pearson correlation or cosine similarity. In this approach, we also limited the number of similar users as k, which makes system more scalable.

- Deep Learning methods: these methods are powerful and they were development due to some limitations of matrix factorization methods like the difficulty of using side features, or the famous problem "cold-start". However they are still under development and there isn't enough material about this approach.

- Matrix Factorization based algorithms: there is a ton of research material on collaborative filtering using matrix factorization. This family of algorithms consists in decomposing the huge, sparse user-item interaction matrix into a product of two smaller and more dense matrices: a user-factor matrix that containing users representations, and a second matrix that multiplies a factor-item matrix that containing items representations. The main idea behind matrix factorization is that exists a pretty low dimensional space of features in which we can represent both users and items and such that the interaction between a user and an item can be obtained by computing the dot product of corresponding dense vectors in that space. The consequence of such factorization is that close users in terms of preferences as well as close items in terms of characteristics, will have close representations in the latent space.

To sum up, the features become extracted features and as they are learned and not given, they taken individually have a mathematical meaning but not intuitive interpretation. However, it is not unusual that the models deriving from these type of algorithms being close to intuitive decomposition that human could think about.

## 2.2 Content-Based methods

The second approach, when designing a recommender system, is *Content-based filtering* method: this method is best suited to situations where there is known data on an item (features), but not on the user.
Content-based filtering methods are based on two things: on a description of the item and on a profile of the user's preferences.

Unlike collaborative methods that, as we said, only depend on the user-item interactions, content based approaches use additional information called "features" about users and/or items.

The principal idea of content-based approaches is to build a model in order to explain the user-item interactions with the help of the available "features".
The model is provided with content that define the representation of users and/or items, usually, users are represented by features and we try to model for each item the type of user profile that likes or not this specific item. Here, as for model based collaborative approaches, a user-item interactions model is assumed. However, this model is more constrained because representation of users and/or items are given.

In these methods, the recommendation problem becomes a classification problem i.e. predict if a user will "like" or not an item or into a regression problem i.e to predict the rating given by a user to an item. In both cases, the model will be based on the user and/or item features at disposal (the "content" of our "content-based" method).

The classification or regression problem given from the recommendation problem can be divided into two categories, based on what are the features are based-on.

If the classification or regression is based on users features, this approach can be called *item-centered*: optimizations, modelling and computations can be done "by item". In this case, a model is built and learnt for a single item on users features trying to find the probabilities of different users to like a specific item.
The model associated with each item is trained on data related to this item and it leads, in general, to robust models as a lot of users have interacted with the item. However, the interactions considered to learn the model come from every users and even if these users have similar features their preferences can be different.
A typical example of an item-centred classification is the "Item-centred Bayesian classifier": for each item a Bayesian classifier is trained that takes user features as inputs and give as output either "like" or "dislike".

So, to achieve the classification, we need to compute:

$$\frac{\mathbb{P}_{item}(like|user\_features)}{\mathbb{P}_{item}(dislike|user\_features)}$$

that it is the ratio between the probability for a user with his/her features to like a specific item and its probability to dislike it. The ratio of conditional probabilities defines the classification rule, that can be expressed following the Bayes formulas:

$$\mathbb{P}_{item}(like|user\_features) = \frac{\mathbb{P}_{item}(user\_features|like)\times\mathbb{P}_{item}(like)}{\mathbb{P}_{item}(user\_features)}$$

$$\mathbb{P}_{item}(dislike|user\_features) = \frac{\mathbb{P}_{item}(user\_features|dislike)\times\mathbb{P}_{item}(dislike)}{\mathbb{P}_{item}(user\_features)}$$

$$\frac{\mathbb{P}_{item}(like|user\_features)}{\mathbb{P}_{item}(dislike|user\_features)} = \frac{\mathbb{P}_{item}(user\_features|like)\times\mathbb{P}_{item}(like)}{\mathbb{P}_{item}(user\_features|dislike)\times\mathbb{P}_{item}(dislike)}.$$

where $P_{item}(dislike) = 1 - P_{item}(like)$ are data pre-computed, while: $\mathbb{P}_{item}(user\_features|like)$ and $\mathbb{P}_{item}(user\_features|dislike)$ are likelihoods assumed to follow Gaussian distributions. Can be done various hypothesis about the covariance matrices of these two likelihood distributions leading to various well known models.

It can be notice once more that, here, likelihood parameters have to be estimated only based on data (interactions) related to the considered item.

Instead, if the classification or regression is based on items features, the method is called *user-centred*: modelling, optimizations and computations can be done "by user". A model is trained by user based on items features that tries to find the probability for each user to like each item.
Then, it is possible to attach a model to each user that is trained on its data: the model obtained is, more personalized than its item-centred counterpart as it only takes into account interactions from the considered user.
However, most of the time, a user has interacted with relatively few items and, so, the model obtained is a far less robust than an item-centred one.

So, if we denote M the user-item interaction matrix, we can create a matrix X where the row vectors represent users coefficients to be learned and we create another matrix Y, where row vectors represent items features that are given.
Then, for a given user "i", we learn the coefficients in $X_i$ by solving the following optimization problem:

$$X_i = argmin_{X_i} \tfrac{1}{2} \textstyle\sum_{(i,j)\epsilon E} [(X_i)\,(Y_j)^T - M_{ij}] + \tfrac{\lambda}{2} (\textstyle\sum_2 (X_{ik})^2)$$

where, "i" is fixed and, so, the first summation is only over (user, item) pairs that concern user "i".
We can observe that if we solve this problem for all the users at the same time, the optimization problem is exactly the same as the one we solve in "alternated matrix factorization" when items are fixed.
This observation marks a link between collaborative filtering and content based approach: model based collaborative filtering approaches (such as matrix factorization) and content based methods both assume a latent model for user-item interactions, but model based collaborative approaches have to learn latent representations for both users and items, while content-based approaches build a model using human-defined features for users and/or items.

## 2.3 Hybrid and other Recommendation Systems

The third and the most approach used nowadays in recommender systems, uses a hybrid approach, combining collaborative filtering and content-based filtering.

Hybrid approaches can be implemented in several ways: by making content-based and collaborative-based predictions separately and then combining them, or by adding content-based capabilities to a collaborative-based approach (and vice-versa), or by unifying the approaches into one model.

Naturally, this method tries to use the advantages of a method to fix the disadvantages of the other.

For instance, as explained above, CF methods suffer from new-item problems, i.e., they can not recommend items that have no ratings, but this problem does not affect content-based approaches, since the prediction for new items is based on their description (features), that are typically easily available.

Several studies have been done that empirically compare the performance of the hybrid approaches with the pure collaborative and content-based methods, and demonstrated that the hybrid methods can provide more accurate recommendations than pure approaches.

There exist also different types of recommendation system in the literature that are worth mentioning but not to analyze deeply.

- Demographic Recommendation System: this type of RS recommends items based on the demographic profile of the user. The main idea is that different suggestions should be predicted for different demographic area. Many web sites adopt effective personalization solutions based on demographics like suggestions customized according to the age of the user.

- Knowledge-based recommendation systems: this method recommends items based on specific domain knowledge, i.e. how certain item features meet users needs and preferences. Knowledge-based systems tend to work better than others at the beginning, but if there is not a learning part in their component, they may be surclassed by other shallow methods that can exploit the logs of the human/computer interaction like in Collaborative Filtering methods.

- Community-based recommendation systems: this method recommends items based on the preferences of the users' friends. Evidence suggests that people tend to trust more on recommendations from their friends than on recommendations from similar but anonymous users. This observation, combined with the explosion of social networks has generated a rising interest in Community Based systems.
  This type of RSs models and acquires information about the social relations of the user and the preferences of the user's friends and recommendations are based on ratings that were provided by the user's friends. In fact these RSs have followed the rise of social-networks and enable a simple and comprehensive acquisition of data related to the social relations of the users.

Although, the most famous methods of Recommender systems are Collaborative Filtering and Content Based methods, among the hybrid methods, there can be also the combination of the above mentioned techniques.

## 2.4 Evaluation and Challenges

Recommendation systems have to afford different challenges: from the problem to find the correct way to evaluate it, or to find the solutions to some of the famous issues that can happens while developing it, like ”cold start”. In the next sections we will first show different ways to evaluate a RS, then we will explain some of the biggest issues that can affect a RS.

### 2.4.1 Evaluation of a recommender system

Recommender systems have different ways of being evaluated. For example, if it is Top-N recommendations (i.e. the most probable items the user will interact with), it is not needed to consider the predictions regarding the rest of the items when conducting the evaluation. However, you could very well be interested in the order of priority of those 5 recommendations, so you would have to consider this. The chosen way of evaluating has an important effect on the way of designing the system.
Two types of recommender system evaluations are frequently discussed: online and offline approaches.

With online methods, users feedbacks are measured given the recommendations made. For example, it can be measured when the user clicks on the recommended items (frequently happens on ads) and evaluate the direct impact of the system. This approach of evaluation is the ideal, although its usually hard to implement because the only way to run the experiments is by interacting with the RS that is already running.

While, the offline methods are ideal for experimental stages, since users aren’t directly involved, and unlike online methods, the system does not have to be deployed. The data is split into training and validation sets, which means that part of the data will be used to construct the system and the other part to evaluate it.

For example, if the recommender system is based on a model that outputs numeric values such as matching probabilities or ratings predictions, it is possible to evaluate the quality of these outputs in a classical way using an error measurement metric. In this case, the model is trained only on a part of the available interactions (train set) and is tested on the remaining ones (test set).
There are a lot of metrics to evaluate performances of a system:

- Mean Absolute Error (MAE)

  Mean absolute error is the average of the difference between the value predicted by a recommender and the actual value given by the user. First, we need to compute the error by subtracting predicted rating and actual rating for each user and then we take the mean of all the errors to calculate MAE:

$$MAE = (\frac{1}{n})\sum_{i=1}^{n}|y_i - x_i|$$

MAE tells how much predicted score is far from the actual score. We take absolute in order to cancel the negative sign, as we are not interested positive or negative score, we only want to know the difference between real and predicted values.
Zero MAE means there was no difference between predicted and actual rating and that the model predicted accurately, therefore, smaller the MAE the better.

- Mean Squared Error (MSE)

  Mean Squared Error is very similar to Mean Absolute Error, but it differs to MAE because, instead of taking absolute of error to cancel the negative sign, it squares it:

  $$MSE = (\frac{1}{n})\sum_{i=1}^{n}(y_i - x_i)^2$$

  MSE helps to penalize results so, even a small difference will result in a big difference. In this case MSE, even if is close to zero means that the recommender system really output good results, because otherwise, the MSE won't be so small.

- Root Mean Squared Error (RMSE)

  As we already said, MSE helps to negate the negative sign but it scales up the errors that can not be compared to actual rating values due to different rating scales. So, RMSE takes the square root of MSE to normalize the scale issue that MSE has.

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

  RMSE is more sensitive to outliers and can exaggerate results if there are outliers in the dataset. Furthermore, RMSE reconstructs the error terms while MAE does not, in fact MAE treats outliers and non-outliers equally while RMSE does not. Also, RMSE will almost always be greater than MAE.

It should be remember that MAE, MSE and RMSE do not have an upper bound, so, that value can not compared with the same metrics from others dataset.

Still, if the recommender system is based on a model that predicts numeric values, there is the possibility to binarize these values with a classical thresholding approach (values above the threshold are positive and values below are negative)

and evaluate the model in classification way.[2]
As the dataset of user-item past interactions can be binary, it is possibile to evaluate the accuracy, the precision and the recall of the binarized outputs of the model on a test set of interactions not used for training[2, 14]. In order to evaluate a model, it is usually take into account the following measures: True Positives (TP): number of instances classified as belonging to class A that truly belong to class A; True Negatives (TN): number of instances classified as not belonging to class A and that in fact do not belong to class A; False Positives (FP): number of instances classified as class A but that do not belong to class A; False Negatives (FN): instances not classified as belonging to class v but that in fact do belong to class A.
The commonly used measure for model performance is: Accuracy. It is defined as the ratio between the instances that have been correctly classified and the sum of instances:

Accuracy = (T P + TN)/(T P + TN + FP + FN). However, accuracy might be misleading in many cases. If we imagine a 2-class problem in which there are an strongly imbalanced of data: a lot of samples of class A and few samples of class B. If a classifier simply predicts everything to be of class A, the computed accuracy would be of very high but the model performance is questionable because it will never detect any class B examples.[14] Other common measures of model performance, particularly in Information Retrieval, are Precision and Recall. Precision is defined as *P = T P/(T P + FP)*, and it is a measure of how many errors we make in classifying samples as being of class A. On the other hand, Recall, *R = T P/(T P + FN)*, measures how good we are in not leaving out samples that should have been classified as belonging to the class.

It must be emphasized that these two measures are misleading when used in isolation in most cases. We could build a classifier of perfect precision by not classifying any sample as being of class A.

At same time, we could build a classifier of perfect recall by classifying all samples as belonging to class A. Indeed, there is a measure, called the F1-measure that combines both Precision and Recall into a single measure as:

$$F1 = \frac{2PR}{P+R}$$

.

At this stage, it is simple to compute *Precision@k*, *Recall@k*, *F1-score@k*, where k is a threshold, commonly known as cut-off rank, where the model is only assessed by considering only its top-most queries. These measures are commonly called *P@k*, *R@k* and *F1@k*.
Another useful metric is the *MAP@k* that stands for mean Average Precision. Firstly, we need to compute the average precision *"AP"* at an arbitrary threshold k of each query of each relevant item where recall increases. This would give a better measurement of our model in its ability to sorting the results of the query.

$$AveragePrecision = \frac{\sum_r P@K}{k}$$

At the end, MAP is Average Precision across multiple queries/rankings and it provides a single-figure measure of quality across recall levels for a set of queries:

$$MAP@ = \frac{1}{N}\sum_{1}^{n} AP_i$$

Finally, if it is considered a recommender system not based on numeric values, that only returns a list of recommendations, like user-user or item-item that are based on a knn approach, it can still be define a precision like metric by estimating the proportion of recommended items that really "suit" the users under investigation. To estimate this precision, it is necessary to exclude items that a specific user has not interacted with and to consider items from the test set for which it has a feedback.

### 2.4.2 Challenges in Recommender System

As we already said, RSs have been deeply studied during recent years, however, they suffers from problems and challenges, and need solutions to work well[14].

- Cold Start problem: the cold start problem is the main issue in the RSs. It arises mainly when a new user or a new item is added to the system. When there is a new user, it is very hard to recommend items if the system doesn't know his/her interests. A solution is to ask to user some preferences about items. If a new item is added, the system will not suggest it to others users because no one rate this specific item before.

- Sparsity Problem: Sparsity Problem is an important issue in recommender systems. This problem is also called "Data sparsity": it means the phenomenon of not observing enough data in a dataset. It happens when a user has a large matrix with a lot of different items but he/she didn't rate most of these items. Exists also two metrics to identify in a better way datasets: "sparsity" that is the number of zero-valued elements divided by the total number of elements and the "density" that is the number of nonzero-valued elements divided by the total number of elements.

- Scalability: Scalability measures the ability of a system to work effectively with high performance while the information is growing. Recommender system needs to recommend items to the users without no problem while the number of users increased or the number of items increased too. To achieve this, it is needed more computations and the RSs get more expensive.

- Gray Sheep: this issue happens when a RS is a collaborative filtering system. Gray Sheep occurs when the ratings/tastes/opinions of a user do not equate to any other group. As a results, the RS can not give to this specific user recommendations.

- Privacy: privacy is one of the important challenges in recommender systems. Recommender systems in order to suggest items that match users' interests need to know some informations. Therefore, users must know or must be informed by companies which datas the RS uses and how it applied. Just think that, the privacy problem is one of the biggest problems of the RSs with the example of the lawsuit that canceled the famous "Netflix Prize". In 2007, two researchers from the University of Texas at Austin were able to identify individual users in the "Netflix Prize Dataset", used in the competition.

There are still many challenges that affects RSs like: Over Specialization Problem, Diversity problem or Novelty problem.
These problems relate to the items the recommendation system suggests. Suggested items to users don't always have to be the same (Novelty) and should not be come all from the same cluster (Diversity).
Over Specialization Problem happens when, as the name suggest, items suggested are too specific because they are based on those already known without discovering new items.
Another important issue is the Shilling Attacks. This problem happens when a malicious user into a system and starts giving false rating, low rating or high ratings in order to decrease or increase popularity on a single one or on a group of items.

# Chapter 3

# ClustKnn

ClustKnn, as the paper published by Karypis et Al, is a scalable collaborating filtering algorithm, and following our description in the second chapter it is a hybrid approach: it combines a model-based approach with a memory based approach and, naturally it tries to take advantages of both methods[11, 12].

A memory-based approach such as User-based KNN uses the entire dataset of user preferences when computing recommendations. These type of algorithms tend to be simple to implement and development, and require a minimum training time. However, their performance tend to be slow as the size of the datasets grow, which makes these algorithms, as stated in the literature, not suitable in large systems. A first workaround could be to only consider a subset of the preferences in the computation, but doing this can reduce recommendation quality. A second workaround could be to perform as much of the computation as possible in an offline setting. However, this may make it difficult to add new users to the system, which is a necessity of most online systems, furthermore, the storage requirements for the precomputed data could be high. On the other hand, a model-based algorithm, such as one based on Bayesian networks computes a model of the preferences/ratings and uses it to compute recommendations. Often, the process of building a model is time-consuming and consequently it is only done periodically. The advantage is that the models are compact and can generate recommendations very quickly, but the disadvantage of model-based algorithms is that if it is added new users, items, or preferences/ratings can be necessary to recompute the entire model.

The first objective that ClustKnn obtained is that, this is very simple and intuitive. It achieved this characteristics utilizing a partitional clustering algorithm for modelling users.

To compute recommendations from the learned model, they used a nearest-neighbour algorithm. However, recommendations can be generated quickly since the data is greatly compressed after the model is built. This, as they said, solved the scalability challenge discussed in the previously chapter.

One interesting property of ClustKnn is its tunable nature. We show later that a

tunable parameter, the number of clusters, can be adjusted to trade off accuracy for time and space requirements.
This makes ClustKnn's structure adaptable to systems of different sizes and allows it to be useful throughout the life of a system as it grows.

The algorithm has two phases and it follows these steps:

- Model-building

  - Select the number of clusters k considering the effects on the recommendation accuracy.
  - Perform Bisecting k-means clustering on the user data. Furthermore, ClustKnn can use any partitional clustering techniques in this stage[5].
  - Build the model with k vector, called "surrogate users", derived from the k centroids: $\{c_1, c_2, ..., c_k\}$. Each $c_i$ is a vector of size m, where m is the number of items, so $c_i = (\widetilde{R}_{c_i,a_1},\ \widetilde{R}_{c_i,a_2},\ ...,\ \widetilde{R}_{c_i,a_m})$, where $\widetilde{R}_{c_i,a_j}$ is the element in the centroid vector that corresponds to item $a_j$. Further, since $\widetilde{R}_{c_i,a_j}$ is an average value of the ratings $a_j$ of all the users in cluster $c_i$, it will be 0 if nobody in that cluster rated $a_j$.

  We can notice that this first step based on clustering is model-based, and it can be computed offline on the training set while the next part is computed online.

- Prediction Generation
  In order to compute the rating of a specific pair (user $u_t$, item $a_t$), the algorithm follows these steps:

  - a user-based CF method: compute the similarity of the target user $u_t$ with each of the surrogate-user using the Pearson correlation coefficient; it must be noted that this computation has to be done only with surrogate model users who have rated $a_t$ otherwise the coefficient will be uncomputable. Adapting the Pearson correlation coefficient with the actual names, we have:

    $$w_{u_t,c_i} = \frac{\sum_{a\epsilon I}(R_{u_t,a} - \overline{R}_{u_t})(\widetilde{R}_{c_i,a} - \overline{R}_{c_i})}{\sqrt{\sum_{a\epsilon I}(R_{u_t,a} - \overline{R}_{u_t})^2 \sum_{a\epsilon I}(\widetilde{R}_{c_i,a} - \overline{R}_{c_i})^2}}$$

    where $I$ is the set of items rated by both the target user $u_t$ and i-th surrogate user.
  - Find and select up to "l" surrogate-users most similar to the target user $u_t$. We tuned this parameter $l$ for each recommendation/prediction.
  - Generate predictions using the *Adjusted weighted average* formula. This computation takes into account the varying degrees of importance of the correlations computed.

    Computing a weighted average, each number in the data set is multiplied by a predetermined weight before the final calculation is made. A weighted average can be more accurate than a simple average in which all numbers are assigned an identical weight:

$$\widehat{R}_{u_t,a_t} = \overline{R}_{u_t} + \frac{\sum_{i=1}^{l}(\widetilde{R}_{c_i,a_t} - \overline{R}_{c_i})w_{u_t,c_i}}{\sum_{i=1}^{l} w_{u_t,c_i}}$$

where $\widetilde{R}_{u_t,a_i}$ is the predicted rating for the user $u_t$ and item $a_i$. As we can see, each correlation $w_{u_t,c_i}$ is weighted by the difference of the value $a_t$ of the surrogate user $c_i$ and the average of all his/her values.

## 3.1 Datasets

Karypis et Al published their results using two different datasets from MovieLens. MovieLens is itself a web-based recommender system and virtual community that recommends movies for its users to watch, based on their film preferences using a collaborative filtering system.
It contains about 11 million ratings for about 8500 movies. MovieLens was created in 1997 by GroupLens Research, a research lab in the Department of Computer Science and Engineering at the University of Minnesota, in order to gather research data on personalized recommendations. Furthermore, GroupLens Research, a human-computer interaction research lab at the University of Minnesota, provides the rating data sets collected from MovieLens website for research use.
There are different datasets on the website, and they were collected over various periods of time, depending on the size of the set. The full data set contains 26,000,000 ratings and 750,000 tag applications applied to 45,000 movies by 270,000 users.
Karypis et Al developed and analyzed ClustKnn using two different datasets:

- **MovieLens 1M Dataset**: this dataset is publicly available, furthermore it contains 1,000,209 ratings of 3,706 movies made by 6,040 users.

- **MlCurrent dataset**: this is a custom dataset created by the authors taking the latest 3 millions ratings and the corresponding users and movies from the dataset more updated at that time.

As we can notice, MlCurrent dataset is irreproducible because it had been created following the rule "the latest 3 million ratings" from the main dataset but in a specific date and we don't know it.

In this thesis we only worked using the Dataset MovieLens 1M (Shortly ML1M), which contains 1.000.209 ratings of 3.706 movies made by 6.040 users.
**MovieLens 1M**
ML1M have been released in February 2003 and it contains 3 files, ratings.dat, users.dat and movies.dat. For the purposes and the analysis of this thesis we only needed "ratings.dat".

The file "ratings.dat" containts all the ratings in the following format:
UserID::MovieID::Rating::Timestamp

where:

- UserID: this ID shows what user has rated a specific movie

- MovieID: this ID shows what movie havs been rated

- Rating: this is the value assigned by the UserID to the MovieID. It can be an integer from 1 to 5.

- Timestamp: the timestamp is represented in seconds since the epoch as returned by time.

Furthermore in this file each user represented by an UserID has rated at least 20 movies.
This information is very important during the computation and the presentation to users of his/her Top-N recommendations since "N recommendations" can not be greater of the number movies rated.
Furthermore, this dataset has a sparsity of 95.53%. As we already said in the paragraph "Challenges in Recommender System", Sparsity of a dataset is defined as the percent of empty cells in the (user, movie) matrix.

The File "users.dat" contains users information and it is in the following format:
UserID::Gender::Age::Occupation::Zip-code

MovieLens ensures that all demographic informations are provided voluntarily by the users and only users who have provided this informations are included in this dataset.

The file "movies.dat" presents movies informations and is in the following format:

MovieID::Title::Genres

where, MovieID matches, naturally, with MovieID presents in "ratings.dat", "Title" is the same title provided by the IMDB (Internet Movie Database), and "Genres" is a pipe-separated value and can be selected from 17 different genres.

Another interesting property where we should focus on are the average rating through all the rating that is 3.58 and the rating distribution.
We can observe all this data on the table below.

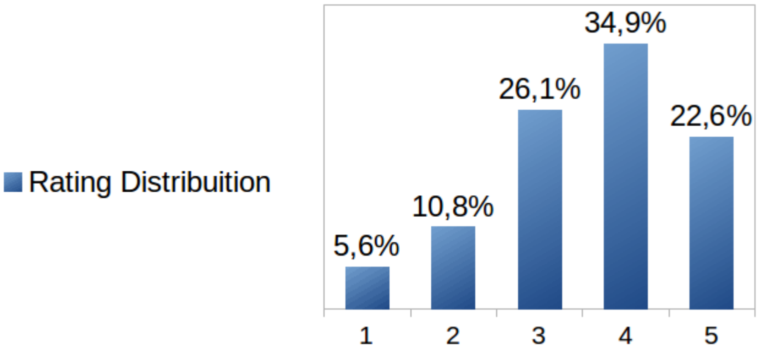


Figure 3.1: Rating distribution of ML1M

| **Property** | **ML1M** |
|---|---|
| Number of users | 6040 |
| Number of movies | 3.706 |
| Minimum $\|u_i\|$, $\forall$ i | 20 |
| Total Number of ratings | 1.000.209 |
| Sparsity | 95.5% |

Table 3.1: Dataset ML1M

## 3.2 Clustering methods

In this section we briefly analyze two different clustering methods that can be used in the model-building during ClustKnn.
Karypis et Al. suggested, and also used for their studies and results Bisecting k-means algorithm, an algorithm that is an improvement of the naive K-means algorithm.
Bisecting k-means algorithm starts by considering all data points as a single cluster, then it repeats the following steps $k-1$ times in order to produce exactly k clusters:

1. Find the largest cluster to split;

2. Apply the basic k-means (2-means) clustering, to generate 2 clusters.

3. Repeat step 2 for j times and take the best split. The best split can be determined following different methods, one way could be of determining which is the best intra-cluster similarity.

The time-complexity of the basic k-means is reported to be $O(n)$ assuming the cost of computing the similarity between the data points as a constant. However, in ClustKnn, this cost is *O(m)* where m is the number of items a user has rated, so the k-means time-complexity becomes *O(mn)*. Therefore, as explained by the authors, the complexity of the Bisecting k-means becomes *O((k - 1)jmn)* $\simeq$ *O(mn)*, that correspond to the offline complexity and model-building of ClustKnn.
Furthermore, Karypis et Al decided to use Bisecting k-means because clusters produced by it tends to be of relatively uniform size considering that, in basic k-means, the cluster sizes may vary significantly, generating poorer quality clusters.

The second clustering method that we would like to explain is called *CLUTO* and it is a package published by George Karypis from University of Minnesota, by Department of Computer Science[6].
By definition, CLUTO is a software package for clustering low- and high-dimensional datasets and for analyzing the characteristics of the various clusters. CLUTO offers three different classes of clustering algorithms that operate either directly in the object's feature space or in the object's similarity space. These algorithms are based on: partitional, agglomerative, and graph-partitioning paradigms.

CLUTO provides seven different criterion functions that can be used to drive both partitional and agglomerative clustering algorithms. Most of these criterion functions have been shown to produce high quality clustering solutions in high dimensional datasets, Furthermore, CLUTO provides some of the more traditional local criteria: single-link, complete-link, and UPGMA, that can be used in the context of agglomerative clustering.

CLUTO's algorithms have been optimized for operating on very large datasets both in terms of the number of objects as well as the number of dimensions. These algorithms can quickly cluster datasets with several tens of thousands objects and several thousands of dimensions. Moreover, since most high-dimensional datasets are very sparse, like ML1M, CLUTO directly takes into account this sparsity and requires memory that is roughly linear on the input size. CLUTO's distribution consists of two stand-alone programs, vcluster and scluster, one for clustering and one for analyzing these clusters, as well as, a library via which an application program can access directly the various clustering and analysis algorithms implemented.

The vcluster and scluster programs are used to cluster a collection of objects into a predetermined number of k clusters. The vcluster program treats each object as a vector in a high-dimensional space, and it computes the clustering solution using one of five different approaches. Four of these approaches are partitional in nature, whereas the fifth approach is agglomerative.
While, scluster program operates on the similarity space between the objects and can compute the overall clustering solution using the same set of five different approaches.

Both the vcluster and scluster programs are invoked by providing two required parameters on the command line along with a number of optional parameters:

- *vcluster [optional parameters] MatrixFile NClusters*

- *scluster [optional parameters] GraphFile NClusters*

Furthermore, there are 18 different optional parameters that can control how vcluster and scluster compute the clustering solution, and this is one of the reasons why CLUTO is a powerful tool in clustering.

## 3.3 Evaluation, Improvements and Results

ClustKnn has been evaluated by their authors using different metrics MAE and NMAE for the rating-prediction quality, while Precision and F1-score for Comparison of top-N recommendation quality[2].

We already introduced *MAE*, while NMAE is a measure of how good a CF algorithm MAEs are over purely random guessing. The Normalized Mean Absolute Error (NMAE) is computed by dividing the MAE of a CF algorithm with the

expected MAE from random guessing.
Formally:

$$NMAE = MAE/E[MAE]$$

Since the ML1M dataset has a rating scale of 1-5 and assuming both ratings and predictions are generated by a uniform distribution:

$$E[MAE] = \frac{1}{25}\sum_{i=1}^{5}\frac{1}{25}\sum_{j=1}^{5}|i-j| = 1.6.$$

On the other hand, the comparison of top-N recommendation quality requires to binarize the output using a simple threshold. The authors consider the target user's relevant items known from the dataset as the ones $u_t$ rated 4.0 or above. Furthermore, since our experiment framework involves dividing the data into training and test sets, they focused on the test set to find the relevant items of the target user $u_t$ and to compute the top-N list for that specific user.
Specifically, the top-N list only contains items that are in the target user's test set. In a similar way, a list of relevant items are also constructed for the target user from his/her test set items. So, based on the relevant list of and the top-N list for the target user, are compute precision-recall-F1 metrics as usual computation.

In the table 3.2 and in the table 3.3 we report the results obtained by Karypis et Al. in their paper, using Bisecting K-means where K = 200. Authors also compared their results with other famous CF algorithms in order to investigate how ClustKnn is performing.

They compared ClustKnn with:

- Singular Value Decomposition (SVD) is a matrix factorization technique that produces three matrices. It uses a matrix structure where each row represents a user, and each column represents an item. The elements of this matrix are the ratings that are given to items by users. So, given the rating matrix A:

  $$SVD(A) = U * S * V^T.$$

  The matrices U, S, and V can be reduced in order to build a rank-k matrix X, $X = U_k * S_k * V_k^T$ that is the closest approximation to the matrix A. SVD requires a complete matrix to operate however, a typical CF rating matrices is very sparse, for example ML1M has 95.5% empty values. To get around this limitation of CF datasets, several solutions have been proposed[19].
  A solution that has been proposed is using average values in the empty cells of the rating matrix[16]. An alternative method, is to find a model that maximizes the log-likelihood of the actual ratings by an Expectation-maximization (EM) algorithm [19]. The EM procedure is rather simple and

is stated below:
Expectation-step: the Missing entries of A are replaced with the values of current X. This creates an expected complete matrix A'.
Maximization-step: Perform SVD(A'). This creates an updated X. Furthermore, the EM process guarantees to converge. The final X represents a linear model of the rating data, and the missing entries of the original matrix A are filled with predicted values.

- User-based KNN: This algorithm belongs to the memory-based class of CF algorithms, as we already talked about in the previous chapter. Predictions under this algorithm are computed as a two step process. First, the similarities between the target user $u_t$ and all other users who have rated the target item $a_t$are computed[3]. That is:

  $$w_{u_i,u_t} = \frac{\sum_{a \epsilon I}(R_{u_t,a} - \overline{R}_{u_i})(R_{u_i,a} - \overline{R}_{u_i})}{\sqrt{\sum_{a \epsilon I}(R_{u_i,a} - \overline{R}_{u_i})^2 \sum_{a \epsilon I}(R_{u_i,a} - \overline{R}_{u_i})^2}}$$

  where $I$ is the set of items rated by both of the users. Then the prediction for the target item $a_t$ is computed using at most $k$ closest users found from step one (k-nearest neighbors), and using a weighted average of deviations from the k users' means:

  $$\widehat{R}_{u_t,a_t} = \overline{R}_{u_t} + \frac{\sum_{i=1}^{k}(R_{u_i,a_t} - \overline{R}_{u_i})w_{u_i,u_t}}{\sum_{i=1}^{k} w_{u_i,u_t}}$$

  is possible compute the rating prediction of an item.

- Item-based KNN: this algorithm is an instance of a memory-based approach too. Predictions are computed by first computing item-item similarities. There can be used different types of similarities, i.e. the adjusted cosine measure for estimating the similarity between two items fits well to this problem:

  $$w_{a,b} = \frac{\sum_{a_i \epsilon U}(R_{u_i,a} - \overline{R}_{u_i})(R_{u_i,b} - \overline{R}_{u_i})}{\sqrt{\sum_{a_i \epsilon U}(R_{u_i,a} - \overline{R}_{u_i})^2 \sum_{a \epsilon U}(R_{u_i,b} - \overline{R}_{u_i})^2}}$$

  Where, U is the set of users who have rated both "a" and "b". When the item-item similarities are computed, the rating space of the target user $u_t$ is examined to find all the rated items similar to the target item $a_t$. In order to generate predictions, it is used a weighted average:

  $$\widehat{R}_{u_t,a_t} = \frac{\sum_{all\_similar\_items,b}(w_{a_t},b * R_{u_t},b)}{\sum_{all\_similar\_items,b}(|w_{a_t},b|)}$$

- Probabilistic Latent Semantic Analysis (pLSA) for collaborative filtering is an elegant generative model proposed by Hofmann et al. [4]. pLSA is a three-way aspect model adapted from their earlier contribution of two-way aspect models applied to text analysis.

  The main idea of pLSA is the notion of the latent class variable Z. The number of states of Z is an input to the model, and each state Z can be interpreted as a different user-type, so $Z = z_1, z_2, ..., z_k$.

Each user belongs to these user-types with a unique probability distribution P(z—u).

This approach models the probability density function *p(r—a, z)* with a Gaussian mixture model and uses an Expectation Maximization (EM) method to learn mixture coefficients *P(z—u)* and p(r—a, z). In the end, the learned model includes P(z—u)s for each user and for each state of Z, and values of $\mu$ and $\sigma$ for each item and each state of Z.

The prediction generation for an item given a pair (user, item) is simply the weighted average of the means of $a_t$ for each state Z.
That is:

$$\widehat{R}_{u_t,a_t} = \sum_z P(z|u_t)\mu_{a_t,z}$$

- Personality Diagnosis is a probabilistic an hybrid CF algorithm: it belongs to model-based and memory-based approaches[10]. This method involves that each user is assumed to have a personality type that captures their true, internal preferences for items. However, the true personality type is not observable, since users rate items by adding a Gaussian noise to their true preferences on the items.

  The probability that: rating of the target user $u_t$ on an item $a_t$ is a value x, given $u_t$ and $u_i$'s personality types are same.

  $$P(R_{u_t,a_t} = x|type_{u_t} = type_{u_i}) = e^{-(x-R_{u_i,a_t})^2/2\sigma^2}$$

  So, the probability that two users' personalities are of the same type is:

  $$P(type_{u_t} = type_{u_i}|R_{u_t}) = \frac{1}{n}\prod_{a\epsilon I} P(R_{u_t,a} = x_a|type_{u_t} = type_{u_i})$$

  where $R_{u_t}$ is the set of ratings of the target user. Finally, the prediction on the target item $a_t$ for $u_t$ is computed as:

  $$\widehat{R}_{u_t,a_t} = argmaxP(R_{u_t,a_t} = x|R_{u_t}) = argmax\sum_i P(R_{u_t,a_t} = x|type_{u_t} = type_{u_i})$$

The metrics computed for prediction qualities of the ratings produced are: MAE and NMAE.

| **CF algorithm** | **MAE** | **NMAE** |
|---|---|---|
| SVD | 0.69 | 0.43 |
| User-based KNN | 0.70 | 0.44 |
| Item-based KNN | 0.70 | 0.44 |
| **ClustKnn (k=200)** | **0.72** | **0.45** |
| pLSA | 0.72 | 0.45 |

Table 3.2: Results from the paper published by Karypis et Al.

They also presented a comparative comparison of the top-N recommendation quality of the selected CF algorithms using Precision and F1 metrics.

| **CF algorithm** | **top-3** | | **top-10** | |
|---|---|---|---|---|
| | **Precision** | **F1** | **Precision** | **F1** |
| SVD | 0.8399 | 0.379 | 0.7564 | 0.6131 |
| User-based KNN | 0.833 | 0.379 | 0.750 | 0.610 |
| Item-based KNN | 0.819 | 0.374 | 0.749 | 0.610 |
| **ClustKnn (k=200)** | **0.825** | **0.377** | **0.743** | **0.606** |
| pLSA | 0.817 | 0.375 | 0.739 | 0.604 |

Table 3.3: Results from the paper published by Karypis et Al.

Here we noticed an inconsistency about the F1-score metric and what the authors wrote on the paper.
They wrote *"Note that more than 50% of the users have only 12 or fewer relevant items in the test sets of Ml1m[...]. Therefore, recall values quickly ramp up and higher values of N provide less valuable information if we want to compare the algorithms.[...]"*.. Again they also added, as we did, that their empirical investigation involved a fivefold cross-validation approach. In other terms, they randomly partition the data into five disjoint folds and apply four folds together to train the CF algorithm, and use the remaining fold as a test set to evaluate the performance. Naturally, they repeated this process five times for each dataset so that each fold is used as a test set once and the results presented are averages over five folds. The discrepancy we found is the number of relevant items for each user declared. We analyzed the dataset and we found that only 236 users over 6040 have at most 12 relevant items and so, if we try to split in a "pseudo-random" the dataset in order to obtain an equally distribution of this 236 users we could obtain only 189 in the train set and 47 on the test set. Again, the test set should be composed by 1208 users, but if we have only 47 users with at most 12 relevant items, the "50% of 1208" can not be reproduced even if we cheat and build the entire test set using all the 236 users.

This is the main reason that we will not compare F1-scores in my results since what we explained before, but we will report only result not comparable with the previous ones.

## 3.4 Implementation and Improvements

In this project in order to implement ClustKnn we had to introduce a tool and tunable value.
The first tool we introduced is CLUTO, we used CLUTO, as explained in the previous section because it already implements bisecting k-means.

CLUTO needs differents parameters and in order to generate clusters we used the following parameters:

*./vcluster -clmethod=rbr -crfun=i2 -sim=corr -cstype=large [MatrixFile] NClusters*
where:

- ”clmethod” selects the method to be used for clustering the objects. There are various that can be chosen, rbr means that k-way clustering solution is computed by performing a sequence of k-1 repeated bisections. During each step, the cluster is bisected so that the resulting 2-way clustering solution optimizes a particular clustering criterion function and in the end the overall solution is globally optimized.

- ”crfun” selects the particular clustering criterion function to be used in finding the clusters. There are seven different clustering criterion functions provided by CLUTO documentation that can be selected by specifying the appropriate integer value. We decided to use i2 that is the default setting for the rb, rbr, and direct clustering methods.” Anyhow i2 matches to a specific optimization function:

  $$\text{i2} = maximize \sum_{i=1}^{k} \sqrt{\sum_{v,u \epsilon S_i} sim(v,u)}$$

  where $k$ is the total number of clusters, $S$ is the total objects to be clustered, $S_i$ is the set of objects assigned to the *ith* cluster, $v$ and $u$ represent two objects, and sim(v, u) is the similarity between two objects.

- ”sim” selects the similarity function to be used for clustering. Among cosine similarity, correlation coefficient, using the Euclidean distance and the extended Jaccard coefficient.

  CLUTO’s clustering algorithms implemented by vcluster treat items to be clustered as vectors in a high-dimensional space and measure similarities between objects using either the cosine function, the Pearson’s correlation coefficient, or a similarity derived from the Euclidean distance of these vectors.

  By using the cosine and correlation coefficient measures, then two objects are similar if their corresponding vectors point in the same direction (i.e., they have roughly the same set of features and in the same proportion, in our case, same two users have seen the same movies), regardless of their actual length. On the other hand, the Euclidean distance does take into account both direction and magnitude. Among these coefficient functions we decided to use the correlation coefficient that best suit for our problem.

- ”cstype” selects the method that is used to select the cluster to be bisected next when -clmethod is equal to ”rb”, ”rbr”, or ”graph”. We decided to bisect to use ”large” as criterion.

A second operation is to make ”l” a tunable value. As we already said, in the prediction generation, in particular when computing the rating:

$$\widehat{R}_{u_t,a_t} = \overline{R}_{u_t} + \frac{\sum_{i=1}^{l}(\widetilde{R}_{c_i,a_t} - \overline{R}_{c_i})w_{u_t,c_i}}{\sum_{i=1}^{l} w_{u_t,c_i}}$$

the authors did not specify how many *surrogate-users* have to be find or taken, they only wrote "up to -l-" so we tried to tune it. So, we fixed $l$ as an integer number from 0 to k, that given the number of cluster similar to the target user $u_t$ we select only $l$ cluster similar. Of course, the similarities must be sorted in descending order and then filtered using this index of reduction $l$.

An improvement we did not implement is the method of *data normalization*. This method is useful when users have not a coherent ways of ratings movies. There can be users that may have rated movies with the range 3-5, or that may have rated all movies with a rating at most 3. We have also used this technique to deal with the "Gray Sheep" problem mentioned in the "Challenges in Recommender System" section. This problem, as we said before, occurs when the tastes / opinions do not match any group and as a result is unable to obtain the benefit of recommendations. In this way, normalizing data, this problem could be in part solved.

However, when a dataset is normalized, in our case, we changed the range of rating from 1-5 (integer) to 0-1.0 in floating point, the entire dataset is completely changed so we can not compare the old results with the new ones.
Noted that, since we could not found the original implementation of ClustKnn we could not tested it using the "*data normalization*" function. Again, this improvement can not be compared with previous results.

In the table below there are the results obtain with our implementation. The tables show the results that take a five fold cross-validation approach over each dataset. Furthermore, we will show the results by the tunable values we mentioned, number of cluster k and number of surrogate-users and each of this combinations by metrics.

| **K-Cluster** | **50** | **100** | **150** | **200** | **250** | **300** |
|---|---|---|---|---|---|---|
| **MAE** | 0,74256 | 0,73598 | 0,73298 | 0,73092 | 0,72916 | 0,72768 |
| **MSE** | 0,91605 | 0,89638 | 0,88676 | 0,88008 | 0,8744 | 0,8697 |
| **RMSE** | 0,95705 | 0,94672 | 0,94162 | 0,93807 | 0,93505 | 0,93253 |

Table 3.4: MAE, MSE, RMSE for K= 50, 100, 150, 200, 250, 300

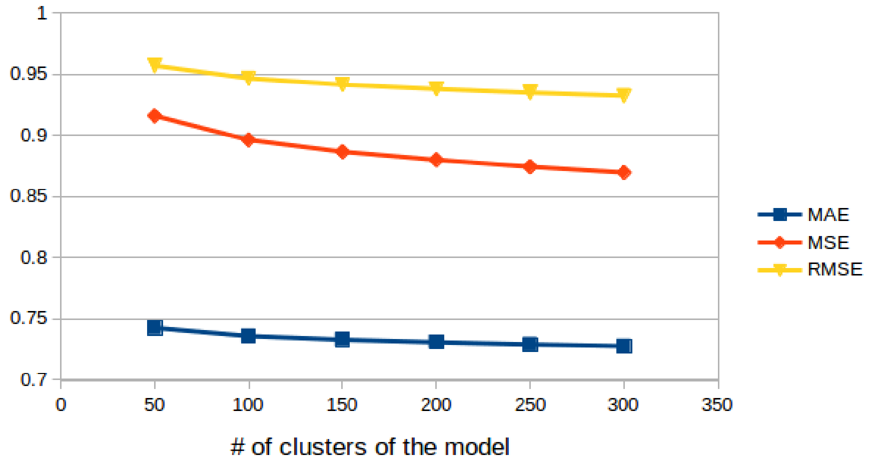


Figure 3.2: Prediction performance based of number of clusters

In the table 3.4, and with the help of the plot in figure 3.2, we can notice that MAE, MSE and RMSE start decreasing more slowly at k=200.
In the table 3.5 we can see the results obtained with number of clusters k = 200 and the number of $l$ maximum compared with Precision@K and F1@K. We used Precision@3, Precision@5, Precision@7 and Precision@10, and the same for F1.

| **# Clusters** | **200** | |
|---|---|---|
| | **Precision** | **F1-Score** |
| **Top-3** | 0,8122 | 0,0976 |
| **Top-5** | 0,79728 | 0,14861 |
| **Top-7** | 0,78132 | 0,19059 |
| **Top-10** | 0,75997 | 0,24194 |

Table 3.5: Results obtained for k = 200

Eventually, if we introduced the tunable parameter $l$: 150, 100, 50, 30, 20. In the tables 3.6 and 3.7 we can see the results. We remember that, $l$ is an integer number from 0 to k, that given the number of cluster similar to the target user $u_t$ we select only $l$ cluster similar. Of course, the similarities must be sorted in descending order and then filtered using this index of reduction $l$.

| **l-reduction** | **150** | | **100** | | **50** | |
|---|---|---|---|---|---|---|
| | **Precision** | **F1-Score** | **Precision** | **F1-Score** | **Precision** | **F1-Score** |
| **Top-3** | 0,81231 | 0,09692 | 0,81545 | 0,09742 | 0,82014 | 0,09782 |
| **Top-5** | 0,79533 | 0,14749 | 0,80003 | 0,14828 | 0,80421 | 0,14878 |
| **Top-7** | 0,77971 | 0,18924 | 0,78356 | 0,19008 | 0,78945 | 0,1911 |
| **Top-10** | 0,75767 | 0,24 | 0,76257 | 0,24135 | 0,76889 | 0,24284 |

Table 3.6: Results obtained for k = 200 and $l$=150, 100, 50.

| **l-reduction** | **30** | | **20** | |
|---|---|---|---|---|
| | **Precision** | **F1-Score** | **Precision** | **F1-Score** |
| **Top-3** | **0,82219** | **0,09797** | 0,82185 | 0,09817 |
| **Top-5** | **0,80778** | **0,14919** | 0,80715 | 0,14926 |
| **Top-7** | **0,79222** | **0,19153** | 0,79127 | 0,19148 |
| **Top-10** | **0,77137** | **0,24318** | 0,77215 | 0,2437 |

Table 3.7: Results obtained for k = 200 and l=30, 20.

We can observe in table 3.8 how much the index $l$ improves the results. Furthermore we compute MAP@5 as an additional metric in table 3.9.

| **l-reduction** | **All** | | **30** | |
|---|---|---|---|---|
| | **Precision** | **F1-Score** | **Precision** | **F1-Score** |
| **Top-3** | 0,80977 | 0,09674 | **0,82219** | **0,09797** |
| **Top-5** | 0,79315 | 0,14715 | **0,80778** | **0,14919** |
| **Top-7** | 0,77689 | 0,18883 | **0,79222** | **0,19153** |
| **Top-10** | 0,75508 | 0,23947 | **0,77137** | **0,24318** |

Table 3.8: Results obtained for k = 200, comparison between l=200 and l=30.

| **l-reduction** | **200** | **30** |
|---|---|---|
| **mAP@5** | 0,85914 | 0,86866 |

Table 3.9: MAP5 for k = 200, comparison between l=200 and l=30.

# Chapter 4

# Our Approach

In our innovative approach we included $Panda^{+}$: a unifying framework for greedy mining approximate top-k binary patterns and their evaluation presented by Orlando et Al. that not substitute the phase of clustering but it offers an improvement[8, 7].
In the following sections we will explain in detail how $Panda^{+}$ works and how we use it in this project.

## 4.1 $PaNDa^{+}$

$PaNDa^{+}$ means discovering *Pa*tterns in *N*oisy *Da*ta. As the name suggests, this algorithm discovers binary patterns in binary data, in a more specific way, it given a binary dataset discovers a set of noise-tolerant, overlapped if extracted, patterns[8, 7]. Data can be expressed in terms of noise $N$ and set of patterns $P^1, P^2, ..., P^n$.

$$P^1 \vee P^2 \vee ... \vee P^n \veebar N = D$$

where:

- $D$: it is a binary dataset;
- $P^n$: it is the representation of a pattern extracted. An approximate pattern extracted from $PaNDa^{+}$ is represented by a pair of sets, items and transactions $(P^n_T, P^n_I)$;
- $N$: it is the noise and it can be expressed it the following way:

  $$N = \vee_{P \epsilon \prod}(P_T * P_I) \veebar D$$

  where $\prod$ is the set of patterns and D is the dataset.

The problem, given $\prod$ and $D$, is find the best set of k patterns $\prod_k$ by minimizing a cost function:

$$J(\prod, D) = ||N||$$

The problem, just exposed, is NP-complete by reduction to the set base problem and it can not be approximated in polynomial time. So, this is the reason why $PaNDa^{+}$ talks about a greedy algorithm for patterns extractions.
The $PaNDa^{+}$ algorithm iterates at most k times, following a 2-stage process:

1. extract a dense core, i.e. a discover a noise-less pattern that covers the yet uncovered 1-bits of the dataset D;

2. extend the dense core to form a good approximate pattern, in order to allow some noises in it;

At each iteration, the pattern that best optimizes the given cost function is added to the set $\prod$ of patterns of solution. This is repeated until k patterns have been found or until it is not possible to improve the cost function.

This method has three main tunable parameters:

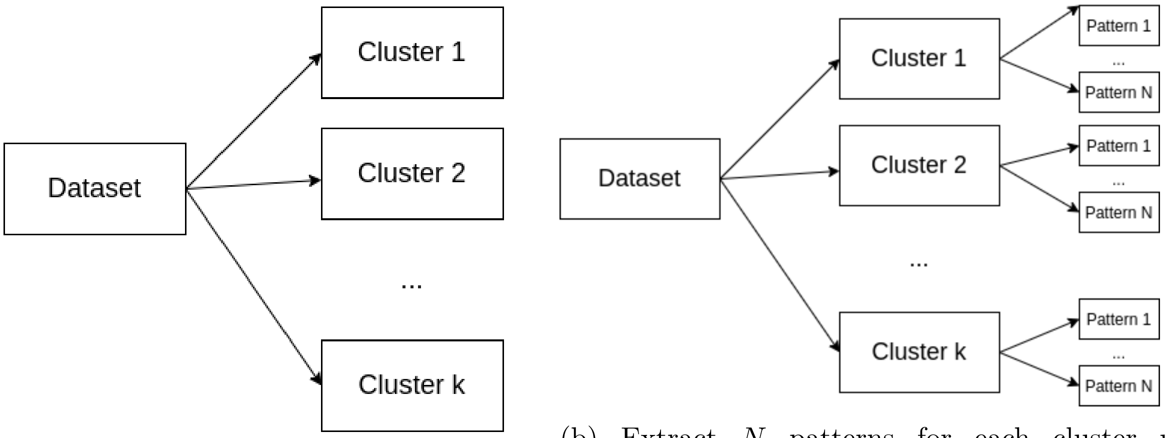
- K: k it is the number of patterns that $PaNDa^+$ tries to extract. $PaNDa^+$ does not guarantee that exactly K patterns will be extracted.

- $\epsilon_r$: max row noise threshold: this parameters is a variable from that has a range from [0,1]. It suggests how much noise it is "allowed" while extending the rows of a dense core. If $\epsilon_r = 0$ means that no noise is allowed, $\epsilon_r = 1$ instead means that, there are allowed some false positives on patterns. When this parameter equals 1, it does not mean that the pattern will be covers the entire dataset analyzed.

- $\epsilon_c$: max column noise threshold: as $\epsilon_r$, this parameter is a tunable parameter by the user from a range 0 to 1. It suggests how much noise it is "allowed" while extending the columns of a dense core. If $\epsilon_c = 0$ means that no noise is allowed, while $\epsilon_c = 1$ instead means that there are allowed some false positives on patterns. When this parameter equals 1 does not mean that the pattern will be covers the entire dataset analyzed.

## 4.2 Main algorithm

The main idea of my algorithm, as we said, is based on $PaNDa^+$, in particular, on the notion of pattern. ClustKnn is based on clusters generated by Bisection K-means or any other algorithms. We tried to add another layer of depth using patterns extracted by $PaNDa^+$ in order to have groups of users more similar.
We applied $PaNDa^+$ not in the whole dataset but in the clusters generated by Cluto.



(a) Find k cluters using Cluto

(b) Extract $N$ patterns for each cluster using $PaNDa^+$

Figure 4.1: First two speps of the model building.

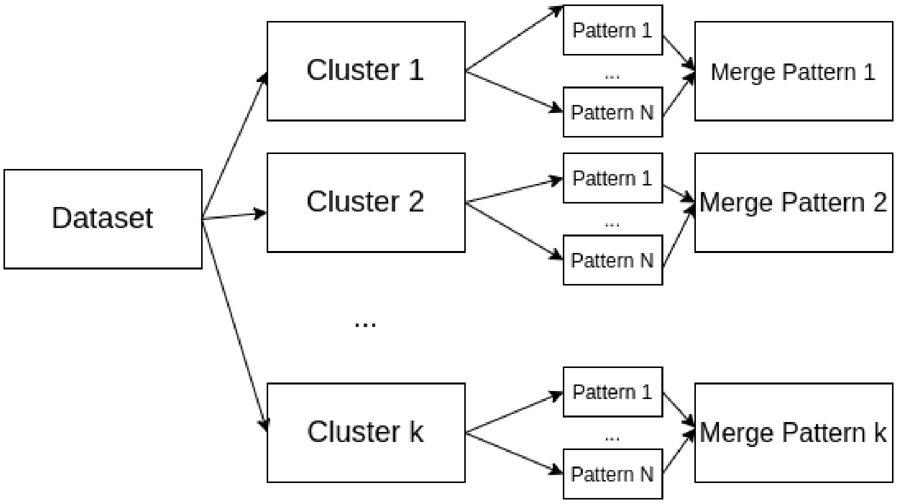


Figure 4.2: Final structure: select N patterns and computer a centroid called *"Merge Pattern"*

In this way our approach has still an offline part and an online part. This algorithm, as ClustKnn, is developed and structured into two different stages, the first is the *Model building* and the second is the *Prediction generation*. The model building phase has the main differences between ClustKnn and this algorithm due to $PaNDa^+$ and to the others tunable parameters.

Model-building:

- Select the number of clusters k considering the effect on the recommendation accuracy as ClustKnn
- Perform Cluto "A Clustering Toolkit"[6], on the user data. Furthermore, as they said and as we have noted there is the possibility to use any partitional clustering technique in this stage, figure 4.1a.
- Execute $PaNDa^+$ on each cluster generated by Cluto and extract all the patterns as we can see in figure 4.1b.

- Build the model with k vector, called "surrogate users", derived from the k centroids: $\{c_1, c_2, ..., c_k\}$. Each $c_i$ is a vector of size m, where m is the number of items, so $c_i = (\widetilde{R}_{c_i,a_1}, \widetilde{R}_{c_i,a_2}, ..., \widetilde{R}_{c_i,a_m})$, where $\widetilde{R}_{c_i,a_j}$ is the element in the centroid vector that correspond to the item $a_j$. Further, since $\widetilde{R}_{c_i,a_j}$ is an average value of the ratings $a_j$ of all users in cluster $c_i$, it will be 0 if nobody in that cluster has rated $a_j$.

- For each $c_i$ centroid there is a list of $M$ patterns extracted. Sorting the patterns by the length of rows, pick only the greatest $N$ patterns for each centroid $c_i$.

- Compute for each surrogate user $c_i$ a merge pattern $P_i$ derived from the $N$ patterns and from the users so $\{P_1, P_2, ..., P_k\}$, as in figure 4.2.
  Each $P_i$ is a vector of size $m$, where $m$ is the number of items, so $P_i = (\widetilde{R}_{P_i,a_1}, \widetilde{R}_{P_i,a_2}, ..., \widetilde{R}_{P_i,a_m})$, where $\widetilde{R}_{P_i,a_j}$ is the element in the centroid vector that correspond to the item $a_j$. Further, since $\widetilde{R}_{P_i,a_j}$ is an average value of the ratings $a_j$ of all users in the merge pattern $P_i$, it will be 0 if nobody in that merge pattern has rated $a_j$.

As usual, the above steps are computed offline as a normal model-based method while the next steps have to be computed online.

Prediction Generation

In order to compute the rating of specific pair user, item ($u_t$, $a_t$) the algorithm follows this steps:

- compute the similarity of the target user $u_t$ with each of the surrogate-users using the Pearson correlation coefficient in order to find K clusters most similar to the target user;

- for each surrogate-user found, select its associated merge pattern $P_i$.

- compute the similarity of the target user $u_t$ with each of the merge patterns $P_i$ selected in the previous steps using the Pearson correlation coefficient. Again at this step using a simple threshold it is possible to it is possible to discard the the least correlated patterns in order to find up to "l" patterns most similar to the target user $u_t$.

- given the Pearson correlation coefficient already computed and using the Adjusted weighted average formula, build a list where there either the $c_i$ or corresponding merge pattern $P_i$;

- Generate predictions using the *Adjusted weighted average* formula as ClustKnn;

Now we want to talk more deeply, for each phase and for each step: its specifics, its strengths and weaknesses, the main differences and similarities with ClustKnn and any future improvements.

The biggest innovation in this project compare to ClustKnn is the adding the patterns extracted by $PaNDa^+$. As we already said, $PaNDa^+$ can generate patterns from a binary datasets $D$, represented by a pair of sets, items and transactions

$(P_T^n, P_I^n)$.
We want to remind that ML1M is a dataset composed by 6040 users (rows) and 3.706 movies (columns) and it contains 1.000.209 ratings, where the ratings are a value from 1 to 5.
Then, once Cluto found clusters with the original dataset following the algorithm above We had to turn the dataset into a binary dataset $D$.

So, in order to let $PaNDa^+$ to discover patterns in each cluster, we decided to binarize the entire dataset using a threshold. As ClustKnn we decided to use "4" as threshold. So, all the rates greater or equal to "4" were transposed to "1" while all the rates lower then 4.0 were transposed to "0". In this way, using $PaNDa^+$ we could found for each cluster $c_i$, $N$ patterns with a variable sizes.

Once we imported all the patterns we decided to modify the entire structure of the pattern: a pattern, as we already saw is composed of different rows and columns, and $PaNDa^+$ works following the similarities not only from rows (users), but also from columns (movies that users have seen and have rated with 4 or high).
So, once a pattern is imported we decided to expand the size of the columns, in such way to have the sizes: rows, the original size of the pattern, while columns the entire size of the dataset.

At the end of the transformations all the patterns are computed and stored in vectors $p_{c_i} = p_1, p_2, ..., p_n$, where $c_i$ is the centroid i and $p_n$ is the pattern n. Each $p_i$ is a vector of size m, where m is the number of items of the dataset, so $p_i = (\widetilde{R}_{p_i,a_1}, \widetilde{R}_{p_i,a_2}, ..., \widetilde{R}_{p_i,a_m})$, where $\widetilde{R}_{p_i,a_j}$ Further, since $\widetilde{R}_{p_i,a_j}$ is an average value of the ratings $a_j$ of all users in the particular pattern cluster $p_i$, it will be 0 if nobody in that pattern has rated $a_j$. Finally, as regards the patterns we want to remind that for each clusters we had to decide how many of them import and which ones. We decided to import the biggest ones based on their sizes, of course the only comparable size was the numbers of rows (users) while the columns as said before is a fixed size.

As regards the numbers of patterns for each cluster, we make it a tunable parameter as it is strongly linked to the size of the data. Any how, in our case, we tried different values and we will report the results with 3,5,7 patterns for each cluster.

At this stage, given $N$ patterns for each cluster, we could build a merge pattern $P_i$. A merge pattern is the average rates of all users belonging only to the patterns. It must be noted that when a user is present twice, it is consider only on time. So $P_i = (\widetilde{R}_{P_i,a_1}, \widetilde{R}_{P_i,a_2}, ..., \widetilde{R}_{P_i,a_m})$, where $\widetilde{R}_{c_i,a_j}$ is the element in the merge pattern vector that correspond to the item $a_j$.

In the prediction generation respect to ClustKnn there are two main phases instead of one.
The first phase consists to find the similar clusters to the target user $u_t$. Once all the similarities are found, for each cluster with a similarity greater or equal to

0.1 the corresponding merge pattern is picked. If in this phase, we fix k similar items as fixed value, we will found at most k merge patterns, because there is the possibility that a merge pattern is not correlated at all with a specific user.

While, the second phase works in a similar way as ClustKnn. Our approach, given for each couple cluster $c_i$ and $P_i$ the correlations computed at the steps before build an ad-hoc structure. Using the *Adjusted weighted average* formula:

$$\widehat{R}_{u_t,a_t} = \overline{R}_{u_t} + \frac{\sum_{i=1}^{l}(\widetilde{R}_{c_i,a_t}-\overline{R}_{c_i})w_{u_t,c_i}}{\sum_{i=1}^{l} w_{u_t,c_i}}$$

We proved that if:

$$(\widetilde{R}_{c_i,a_t} - \overline{R}_{c_i})w_{u_t,c_i} > (\widetilde{R}_{P_i,a_t} - \overline{R}_{P_i})w_{u_t,P_i}$$

means that the cluster $c_i$ has more effect in the adjusted weighted average. On the other hand, if:

$$(\widetilde{R}_{c_i,a_t} - \overline{R}_{c_i})w_{u_t,c_i} < (\widetilde{R}_{P_i,a_t} - \overline{R}_{P_i})w_{u_t,P_i}$$

means that the merge pattern has more effect in the adjusted weighetd average. The reason for this choice is also because we realized that only correlations is not enough because in this way both the value of the movie $\widetilde{R}_{P_i,a_t}$ and the average either the pattern $\overline{R}_{P_i}$ or cluster $\overline{R}_{P_i}$ are taken into account.

To sum up, we choose, using $(\widetilde{R}_{c_i,a_t}-\overline{R}_{c_i})w_{u_t,c_i}$ either cluster $c_i$ or the corresponding merge pattern $P_i$. Once, this structure is build, we used the *Adjusted weighted average* as usual.
We want to specify that, the first phase of the prediction generation doesn't assure to find a similarity for each cluster. This can happens in two cases: the first is when the similarity can not be computed: it means that the target user $u_t$ does not have any rated movies in common with that particular cluster, because as we know, in Pearson correlation coefficient formula both the numerator and the denominator are strictly related to $\sum_{a\epsilon I}$ where $I$ is the set of items rated by both the target user $u_t$ and i-th surrogate user.
The second case is when the similarity is less than 0.1: we decided a fixed threshold to eliminate the weaker patterns right away and reduce the number of clusters used in the *Adjusted weighted average* computation.
The same thing is done with the computation of the similarity of the merge patterns, so the similarities less than 0.1 are cut off from the Adjusted weighted average computation.

To sum up, compared to ClustKnn, we had to add some tunable parameters in order to obtain better results.

- k clusters: $k$ is the number of cluster generated by Cluto. This parameter must be chosen in the model building phase.

- N patterns: $N$ is the number of patterns memorized for each cluster $c_i$. Given a list of patterns, the largest $N$ patterns are taken considering the numbers of rows. $N$ is strictly related to the number of clusters $k$ because as $k$ grows the number of users in each clusters decreases, this means that $PaNDa^+$ can extract less patterns. Instead if $k$ decreases, the number of users in each clusters increases and $PaNDa^+$ can extract more patterns.

- l - index of reduction: $l$ is an integer number from 0 to $k$, that given the number of cluster similar to the target user $u_t$ we select only l cluster similar. Of course, the similarities must be sorted in descending order and then filtered using this index of reduction $l$. On the other hand, if we select $l$ clusters, we will select $l$ merge patterns.

We already mentioned the algorithms that Karypis et Al compared with ClustKnn. Now, we want to compare ClustKnn with our personal approach and with the others algorithms. In particolar we want to report the complexity comparison that Karypis et Al. did in their report and add the complexity of our approach.

## 4.3 Comparison and results

As ClustKnn, our algorithm has two main phases, the first is computed offline (model-based) and the other must be computed online (memory-based).

- The model based consists in two different offline phases:
  - the first part is the generation of the clusters by Cluto. The k-means by Cluto is time-complexity the complexity of the Bisecting k-means becomes $\mathcal{O}((k-1)jmn) \cong \mathcal{O}(mn)$, which is the offline complexity of the first part.
  - the second part is the patterns generations. $Panda^+$ is a scalable algorithm, and its computational complexity is linear in the number of transactions $n$ and quadratic in the number of items $m$, so: $\mathcal{O}(knm^2) \cong \mathcal{O}(nm^2)$.

  So the offline phase has a complexity of: $\mathcal{O}(max(nm, nm^2)) \cong \mathcal{O}(nm^2)$

- The online phase consists in also two differents phases:
  - The first phase is find the correlation with the k clusters so $\mathcal{O}(k)$ similarity are computed. Each calculation takes $O(m)$ time, so this stage requires $\mathcal{O}(km) \cong \mathcal{O}(m)$
  - The second step is find the correlation with the k merge Patterns, so $\mathcal{O}(k)$ similarity are computed. Each calculation takes $O(m)$ time, so this step requires $\mathcal{O}(km) \cong \mathcal{O}(m)$

  Instead, the online phase has a complexity of: $\mathcal{O}(m) + \mathcal{O}(m) \cong \mathcal{O}(2m) \cong \mathcal{O}(m)$

The table 4.1 reports the computational complexity of the different algorithms compared by Karypis et Al.

| **CF algorithm** | **Offline** | **Online** |
|---|---|---|
| pLSA | $\mathcal{O}(mn)$ | $\mathcal{O}(m)$ |
| SVD | $\mathcal{O}(n2m + m2n)$ | $\mathcal{O}(m)$ |
| Personality Diagnosis | - | $\mathcal{O}(mn)$ |
| ClustKnn | $\mathcal{O}(mn)$ | $\mathcal{O}(m)$ |
| User-based KNN | - | $\mathcal{O}(mn)$ |
| Item-based KNN | - | $\mathcal{O}(mn)$ |

Table 4.1: Comparison of time-complexities of the selected CF algorithms

In the table 4.2 we can notice a decreasing of the performance in the model build phase. In particular the decrease is due to $PaNDa^+$ extraction patterns. However, due to the great sparsity of the datasets in recommendation systems, in our implementation, we could reduce the time computation used by $Panda^+$ during the implementation removing the transactions (items) that no ones rated from the binary dataset $D_i$.

| **CF algorithm** | **Offline** | **Online** |
|---|---|---|
| ClustKnn | $\mathcal{O}(mn)$ | $\mathcal{O}(m)$ |
| Our approach | $\mathcal{O}(m^2n)$ | $\mathcal{O}(m)$ |

Table 4.2: Comparison of time-complexities of ClustKnn with my approach

In the table below there are the results obtain with our implementation. The tables will show the results that take a five fold cross-validation approach over each dataset. Furthermore, we will show the results by the tunable values we mentioned, number of cluster k, number of surrogate-users, number of patterns considered and each of this combinations by metrics.

| **K-Cluster** | | **50** | **100** | **150** | **200** | **250** | **300** |
|---|---|---|---|---|---|---|---|
| Pattern | Metric | | | | | | |
| **3** | **MSE** | 0,75431 | 0,74594 | 0,74098 | 0,73916 | 0,74219 | 0,74002 |
| **3** | **MAE** | 0,95031 | 0,92729 | 0,91337 | 0,90821 | 0,9165 | 0,91082 |
| **3** | **RMSE** | 0,97475 | 0,96287 | 0,95561 | 0,95293 | 0,95731 | 0,95427 |
| **5** | **MSE** | 0,7534 | 0,74511 | 0,7405 | 0,73884 | 0,74219 | 0,73997 |
| **5** | **MAE** | 0,94774 | 0,92486 | 0,91186 | 0,90711 | 0,91629 | 0,91052 |
| **5** | **RMSE** | 0,97343 | 0,96161 | 0,95482 | 0,95236 | 0,9572 | 0,95411 |
| **7** | **MSE** | 0,7527 | 0,74477 | 0,74022 | **0,7387** | 0,74214 | 0,73993 |
| **7** | **MAE** | 0,94575 | 0,92384 | 0,91096 | **0,90663** | 0,91608 | 0,91037 |
| **7** | **RMSE** | 0,97241 | 0,96108 | 0,95436 | **0,95211** | 0,95709 | 0,95403 |

Table 4.3: MSE, MAE, RMAE with K = 200 and N = 7

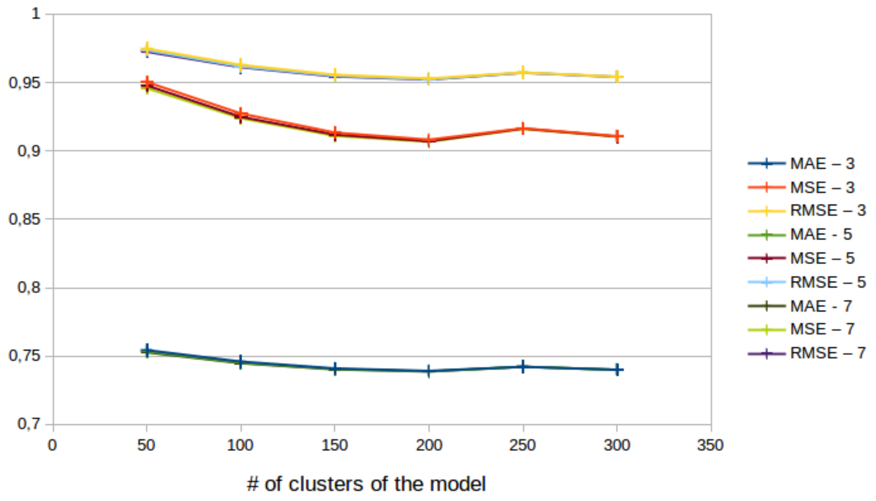


Figure 4.3: MSE, MAE, RMAE

In the table 4.3, we can see the metrics MSE, MAE, RMAE that we already mentioned. We tested the algorithm with different values of $k$ clusters, with the maximum value of $l$ and with different values of $N$ patterns, and with the help of the plot in figure 4.3 we can assert and note explicitly that $k = 200$ and $N = 7$ could give the best results.

In the next tables we will show the final results of precision and F1-scores of our project. In the table 4.4 we can see the results obtained with number of clusters k = 200, with all the surrogate users found and merge patterns found and $N$ = 7 compared with Precision@K and F1@K. We used Precision@3, Precision@5, Precision@7 and Precision@10, and the same for F1. In this case, when we talk about *"all"* surrogate users and *"all"* merge patterns we refer to l=200.

| **# Clusters** | **200** | |
|---|---|---|
| | **Precision** | **F1-Score** |
| **Top-3** | 0,81634 | 0,09805 |
| **Top-5** | 0,80189 | 0,14988 |
| **Top-7** | 0,78763 | 0,19275 |
| **Top-10** | 0,76846 | 0,24594 |

Table 4.4: Results obtained for k = 200, l=200, N = 7.

Introducing the tunable parameter l: 150, 100, 50, 30, 20 we obtained the results showed in tables 4.5 and 4.6. We remember that, l is an integer number from 0 to k, that given the number of cluster similar to the target user $u_t$ we select only l cluster similar. Of course, the similarities must be sorted in descending order and then filtered using this index of reduction l. On the other hand, if we select l clusters, we will select only l merge patterns.

| **l-reduction** | **150** | | **100** | | **50** | |
|---|---|---|---|---|---|---|
| | **Precision** | **F1-Score** | **Precision** | **F1-Score** | **Precision** | **F1-Score** |
| **Top-3** | 0,81915 | 0,09843 | 0,82235 | 0,09885 | 0,82638 | 0,09941 |
| **Top-5** | 0,80358 | 0,15016 | 0,80682 | 0,15066 | 0,81169 | 0,15149 |
| **Top-7** | 0,78964 | 0,19316 | 0,79295 | 0,19389 | 0,79745 | 0,19476 |
| **Top-10** | 0,77058 | 0,24652 | 0,77416 | 0,24742 | 0,77869 | 0,24868 |

Table 4.5: Results obtained for k = 200, N = 7 and l=150, 100, 50.

| **l-reduction** | **30** | | **20** | |
|---|---|---|---|---|
| | **Precision** | **F1-Score** | **Precision** | **F1-Score** |
| **Top-3** | **0,82533** | **0,09891** | 0,82445 | 0,09907 |
| **Top-5** | **0,81242** | **0,15117** | 0,81017 | 0,15097 |
| **Top-7** | **0,79804** | **0,1943** | 0,79808 | 0,19495 |
| **Top-10** | **0,77982** | **0,24801** | 0,77997 | 0,24855 |

Table 4.6: Results obtained for k = 200, N = 7 and l=30, 20.

We can observe in table 4.7 how much the index l improves the results. Furthermore we compute MAP@5 as an additional metric in table 4.8.

| **l-reduction** | **All** | | **30** | |
|---|---|---|---|---|
| | **Precision** | **F1-Score** | **Precision** | **F1-Score** |
| **Top-3** | 0,81634 | 0,09805 | **0,82533** | **0,09891** |
| **Top-5** | 0,80189 | 0,14988 | **0,81242** | **0,15117** |
| **Top-7** | 0,78763 | 0,19275 | **0,79804** | **0,1943** |
| **Top-10** | 0,76846 | 0,24594 | **0,77982** | **0,24801** |

Table 4.7: Results obtained for k = 200, comparison between l=200 and l=30.

| **l-reduction** | **200** | **30** |
|---|---|---|
| **MAP@5** | 0,86313 | 0,87088 |

Table 4.8: MAP@5 for k = 200, comparison between l=200 and l=30.

# Chapter 5

# Conclusion

At the end of this project we can say that even if we can see an increase in computational complexity at the level of the model building, that is, in the offline phase, the results have improved.
This shows not only that the project offers real improvements but the dataset gives all the information and data necessary to provide increasingly accurate predictions and/or results.
Furthermore, if we analyze our approach more deeply we can notice that a merge pattern is nothing more than a cluster with fewer users. This means that using the patterns extracted by $PaNDa^+$ and so a unify pattern is more efficient due to the high quality patterns extracted.

Finally, we must consider the fact that there is a part of users that have less than 10 relevant items, however they are only 90 out of 6040, and this means that any precision@N with N greater than 10, 15 or more can not ideally be at 100%.

For example, when adding data to the dataset it is possible to expect three types of scenarios.
The first case is when the user with the addition of new rates deviates from the merge pattern it belongs but not from the cluster it belongs. Consequently, it must be somewhat displaced from the merge pattern. This can be shifted through the computation of a correlation.

The second case is when the user with the addition of new rates deviates completely from the cluster to which it belongs. This means that a user is added to another cluster and this means that it can modify the merge pattern of that cluster.

The third case is when a new user is added. The ”cold start” problem occurs as usual, however it is possible to present predictions through the closest clusters and merge patterns to the user.

In all three cases, however, it is not necessary to reformulate the model, consequently *Cluto* to generate clusters and *Panda*$^+$ to find patterns must not be executed at each new addition in a possible dataset.

Table 5.1 and table 5.2 summarize the improvements obtained in this thesis. We

| | **ClustKnn** | | **ClustKnn l=30** | | **Our approach l=30** | |
|---|---|---|---|---|---|---|
| | **Precision** | **F1-Score** | **Precision** | **F1-Score** | **Precision** | **F1-Score** |
| **Top-3** | 0,80977 | 0,09674 | 0,82219 | 0,09797 | **0,82533** | **0,09891** |
| **Top-5** | 0,79315 | 0,14715 | 0,80778 | 0,14919 | **0,81242** | **0,15117** |
| **Top-7** | 0,77689 | 0,18883 | 0,79222 | 0,19153 | **0,79804** | **0,1943** |
| **Top-10** | 0,75508 | 0,23947 | 0,77137 | 0,24318 | **0,77982** | **0,24801** |

Table 5.1: Comparison of results obtained by ClustKnn

| | **MAP@5** |
|---|---|
| **ClustKnn l=200** | 0,85914 |
| ClustKnn l=30 | 0,86866 |
| Our approach l=30 | **0,87088** |

Table 5.2: Comparison of results obtained by ClustKnn

# Bibliography

[1] The acm conference series on reccomender system, 2020. URL https://recsys.acm.org/recsys21/.

[2] Chumki Basu, Haym Hirsh, and William Cohen. Recommendation as classification: Using social and content-based information in recommendation. page 714–720, 1998.

[3] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, page 230–237, New York, NY, USA, 1999. Association for Computing Machinery. ISBN 1581130961. doi: 10.1145/312624.312682. URL https://doi.org/10.1145/312624.312682.

[4] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, January 2004. ISSN 1046-8188. doi: 10.1145/963770.963774. URL https://doi.org/10.1145/963770.963774.

[5] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, September 1999. ISSN 0360-0300. doi: 10.1145/331499.331504. URL https://doi.org/10.1145/331499.331504.

[6] G. Karypis. Cluto - a clustering toolkit. 2002.

[7] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Mining top-k patterns from binary datasets in presence of noise. pages 165–176, 04 2010. doi: 10.1137/1.9781611972801.15.

[8] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. A unifying framework for mining approximate top- binary patterns. *IEEE Transactions on Knowledge and Data Engineering*, 12 2014. doi: 10.1109/TKDE.2013.181.

[9] Marwa Mohamed, Mohamed Khafagy, and Mohamed Ibrahim. Recommender systems challenges and solutions survey. 2019. doi: 10.1109/ITCE.2019.8646645.

[10] David M. Pennock, Eric J. Horvitz, Steve Lawrence, and C. Lee Giles. Collaborative filtering by personality diagnosis: A hybrid memory- and model-based approach, 2013.

[11] Al Rashid, Shyong Lam, George Karypis, and John Riedl. Clustknn: a highly scalable hybrid model- & memory-based cf algorithm. 09 2006.

[12] Al Mamunur Rashid, Shyong K. Lam, Adam LaPitz, George Karypis, and John Riedl. Towards a scalable knn cf algorithm: Exploring effective applications of clustering. In Olfa Nasraoui, Myra Spiliopoulou, Jaideep Srivastava, Bamshad Mobasher, and Brij Masand, editors, *Advances in Web Mining and Web Usage Analysis*, pages 147–166, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-77485-3.

[13] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems Handbook*, volume 1-35, pages 1–35. 10 2010. doi: 10.1007/978-0-387-85820-3_1.

[14] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*, pages 257–294. 2011.

[15] Roberto Saracco. A changing world measured in internet time, 2020. URL https://cmte.ieee.org/futuredirections/2020/07/13/a-changing-world-measured-in-internet-time/.

[16] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system – a case study. 08 2000.

[17] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. *Proceedings of ACM World Wide Web Conference*, 1, 08 2001. doi: 10.1145/371920.372071.

[18] Meenakshi Sharma and Sandeep Mann. A survey of recommender systems: Approaches and limitations. 2013.

[19] Nathan Srebro and Tommi Jaakkola. Weighted low-rank approximations. 2, 08 2003.