Master in Artificial Intelligence (UPC-URV-UB)

Master of Science Thesis

An Analysis of Memory Based Collaborative Filtering Recommender Systems with Improvement Proposals

Claudio Adrian Levinas

Advisor: María Salamó Llorente

September 2014

Abstract

Memory Based Collaborative Filtering Recommender Systems have been around for the best part of the last twenty years. It is a mature technology, implemented in numerous commercial applications. However, a departure from Memory Based systems, in favour of Model Based systems happened during the last years.

The Netflix.com competition of 2006, brought the Model Based paradigm to the spotlight, with plenty of research that followed. Still, these matrix factorization based algorithms are hard to compute, and cumbersome to update. Memory Based approaches, on the other hand, are simple, fast, and self explanatory. We posit that there are still uncomplicated approaches that can be applied to improve this family of Recommender Systems further.

Four strategies aimed at improving the Accuracy of Memory Based Collaborative Filtering Recommender Systems have been proposed and extensively tested. The strategies put forward include an Average Item Voting approach to infer missing ratings, an Indirect Estimation algorithm which pre-estimates the missing ratings before computing the overall recommendation, a Class Type Grouping strategy to filter out items of a class different than the target one, and a Weighted Ensemble consisting of an average of an estimation computed with all samples, with one obtained via the Class Type Grouping approach.

This work will show that there is still ample space to improve Memory Based Systems, and raise their Accuracy to the point where they can compete with state-of-the-art Model Based approaches such as Matrix Factorization or Singular Value Decomposition techniques, which require considerable processing power, and generate models that become obsolete as soon as users add new ratings into the system.

Acknowledgements

Artificial Intelligence is a fascinating topic, which certainly will touch our lives in the years to come. But out of the many branches of this rich discipline, Recommender Systems attracted me particularly. This, I owe to the teachings of María Salamó Llorente, who introduced me to the topic, patiently answered all my numerous questions, and after completing the course, was kind enough to agree to supervise my Thesis. I admire her patience and her focus. Without her, this work would be half as interesting and half as useful.

No man is an island. And I would never have made it this far, this advanced in life, without the support of my wife and my son. They are my pillars. They are my ground. They are my light. Little makes sense without them. Thank you both. Thank you for pushing me further and higher.

In the memory of my mother.

Dreams, like faith, might well move mountains.
But reality is very keen on bringing them down.

So, know your limits better than your dreams.
Then, perhaps, you might achieve them one day.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Definition of the Problem

Up until the advent of the internet, shoppers browsing for potential merchandise to purchase would either follow their innate likes and tastes, or follow those of a person they trusted specially. Arguably, we all have a natural trait that compels us to classify things: this we like; this we don't. And in principle, it is possible albeit difficult, to browse all merchandise in a shop or shopping centre, and come out with the one piece that is really worth buying.

The "selection by inspection" paradigm works wonders, as it places a high bar between the things we would like to consider further, and those we can immediately discard. However, there is a fundamental assumption behind the success of this behavioural pattern that bears remembering: it presupposes that the number of items we will sort is manageable. But if the internet has taught us anything in the past twenty years is that information grows exponentially, and we can't read it all.

The limitation in the number of potential items one can look at in the web, is virtually non existent. We have all been presented with hundreds of millions of hits from a Google search, or pages upon pages of images when looking for a pictorial keyword. Information overflow is common place in our age. We have transcended the need to "be there" in order to browse, since the actual physical object depicted or described, is only stored as a bit stream in a server with high accessibility.

During the early years of the internet, while shopping was moving online at a slow pace, the old paradigm of "selection by inspection" seemed to work at first. The physical browsing and window shopping was literally translated into jumping from a rudimentary online shop to another. Still, arguably, a manageable task. There were few products online to choose from, and yet fewer websites with online shops set up.

Things changed significantly with the inception of the Web 2.0 around the year 2004. Everything and everybody moved to the internet, shops and shoppers alike, and radical changes to the GUI of web pages, particularly those enabled by the AJAX responsive technology, permitted a much richer and immersive experience of browsing.

From then on, it was a never ending exponential increase in information flow, that even the physical network layer had problems providing at first. But soon enough the speed limit imposed by early modems gave way to high speed ADSL or cable, and from that moment onward, web pages competed with one-another to display more graphics, more images, and more text. The information age was born, and the concept of information overload became thence common knowledge.

How should an online shopper filter out the massive amount of options being offered now? The task truly became akin to finding a needle in a haystack. Using the old fashion pre-internet method of trusting one's own taste or that of a close friend did not seem to cut it any longer. The visibility of information became greatly reduced. How to discover new hidden things? New products? New ideas? How to select among "literally" millions of images, for the one sought?

Googleing a term and expecting to find a reasonable handful of products is not a sensible expectation any longer. What should the query "I want a shirt like the one I bought last week" return? Search engines are good at returning matches to keywords, but the results are not tailored to the user. Arguably, very particular queries do yield valuable results, however the lists are so vast and devoid of any personal targeting, that in many cases they are rather meaningless.

Luckily, online shops faced the question of targeted recommendation rather early, perhaps realizing that information would only scale with time, and that people would need help to find the items they want. Their quest took them to devise algorithms that could either learn one's taste, or measure the "similarity" between our taste and that of others, and present us with an educated recommendation as to what we might like to browse next.

The underlying premise behind their premise is rather simple but powerful: quantifying and classifying taste can serve as a platform in which other items that shoppers might like, can be offered to them. Clearly, assisting shoppers in browsing experiences where the number of items is just too vast to cover physically, aims at increasing the probability of converting a casual look into a sale.

The family of algorithms developed were termed "Recommender Systems", and rapidly moved from being research projects, into becoming incorporated in full blown commercial products. Amazon.com was one of the first websites to integrate a Recommender System to aid in the discovery of their vast catalogue of books. Today we all expect to be presented with the familiar "customers who bought this item, also bought such and such". Or "these items are generally bought together".

Recommender Systems are a product of research carried out in the field of Information Retrieval, particularly into Information Filtering techniques developed to better cope with the exponential growth of information in the computer age. This places Recommender Systems within the field of Data Mining and Machine Learning. In fact, Recommender Systems are said to "learn" the preferences of a particular user, with the intention of suggesting relevant *not-yet seen* items to this target user.

Modern Recommender Systems nurture from Machine Learning and Statistics, and will employ Natural Language Processing techniques to when analysing the contents of items [4]. They will also leverage inference of user interest, which is a classification problem, in order to build a rich user profile that would represent the target user as faithfully as possible.

One of the earliest research projects on Recommender Systems was GroupLens from 1994 [47], which presented UseNet users with similar news stories to follow and read. The same research group produced MovieLens in 1997 which strived to recommend movies to users, based on the preferences of a community of film aficionados rating their personal preferences.

Another noteworthy project known as the Music Genome Project was launched by Pandora.com in 2000 , consisting on a Recommender System that learnt the similarities between music genres, artists, and songs, and offered tunes matching a user's taste. From the same year is Netflix.com which started providing movie rentals online, and offered movie recommendations via its in-house Recommender System appropriately called Cinematch.

Fast forward to the present date, and one would hardly find an online shop or an online community without a Recommender System running in a background process; learning the user's taste, and working hard to match it to the available list of products on stock or stories on record.

Recommender Systems [48, 26] are today a ubiquitous technology which has proven invaluable when dealing with the exponential growth of information. News, products, music, posts, all can be adapted to take advantage of the algorithm's matching strength and personalized profile. The success of the technology can easily be seen by studying one of Google's most important improvements to their search engine: browsing history tracking. Learn a user's habits at browsing and leverage it to recommend adds for potential website visits.

Much research, both academic and in the industry, has been carried out in the field of Recommender Systems. And while the topic is now twenty years old, the techniques developed remain rather simple. Among the various algorithms developed, two broad categories stand out, one which matches the users' tastes and is known as Collaborative Filtering [52, 18, 26], and another which matches the items' specifications and is known as Content Based [43, 48, 26].

In a Collaborative Filtering Recommender System the main goal is to learn the user's taste by means of some metric, and then match this particular target user to a large database of other shoppers for which their preferences have been obtained in the past. One simple way of quantifying a user's like or dislike for an item is to use a rating. This can take a binary form as in the case of "like" vs "dislike", or have more shades as found in a 1-to-5 star rating scale.

Regardless of how the metric is obtained, the underlying assumption is that a rating is a good enough representation of the true feeling of a user towards a particular item. And once enough ratings have been collected in a community of users, a Collaborative Filtering algorithm can start working to recommend target users other items that might be of interest to them.

Most forms of Collaborative Filtering Recommender Systems work by matching a target user to a large list of candidates, by using a similarity function. This approach is termed User Based [52, 18, 48, 26] since the primary focus of the algorithm is finding similar candidate users to the target user. In contrast to the User Based approach, Recommender Systems can search for similar items to the target item, in which case the algorithm is termed Item Based [52, 18, 48, 26].

The algorithm takes the list of rated items from the target user, matches them to the rated items from a candidate user, and computes the squared error difference or another suitable measure function, where the smaller the result, the closer the candidate's taste to the target user. Any product in the candidate's list, not found in the target user's list can readily be offered as a recommendation, under the assumption that similar tastes will like similar items.

Many different similarity functions have been investigated to match a target user to their fellow candidates. Examples of these functions are the Euclidean Distance, the Pearson Correlation, and the Cosine Distance [44]. They all strive to quantify the separation between two samples of rated items, so that a neighbourhood of similar candidates can be identified for the target user.

In the context of Collaborative Filtering, the Recommender System is said to be Memory Based [44] when the algorithm computes the similarities in-memory without the need to produce a model first. It is also generally termed Neighbourhood Based [48] because it searches for the best candidates within a set neighbourhood.

Collaborative Filtering algorithms saw a powerful alternative formulation emerge when Netflix.com proposed in 2006 a one million dollar prize to the individual or group that could improve "Cinematch" by 10% or better [7, 31, 32]. It took three years for the prize to be finally awarded, something that in itself speaks about the immense difficulty in trying to improve these systems. The feat, in fact, was only achieved when the top contenders joined forces in a collaborative work, close to the finish line, to finally grab the prize that kept slipping away year after year.

This new line of Recommender Algorithms is termed Model Based [18, 48, 26], and makes use of matrix factorization techniques to guess at the values of latent factors, which loosely stated could be thought as underlying descriptors of the domain. For example, in a movie recommender system, latent factors could describe drama vs. comedy, age appropriateness, amount of violence, and are computed by factoring the sparse matrix of users and items ratings.

The idea of using a factoring strategy came up during the Netflix.com competition when one of the contenders described the idea in a blog post. Other participants quickly saw the potential, and put the proposal to work. At the end, the winning team confessed that it relied heavily on matrix factorization to achieve the 10% improvement goal set by the judges.

While matrix factorization does currently have an upper hand on Recommender Systems, its benefits don't come for free. For starters, the algorithm is based on factoring an inherently sparse matrix, which can easily be proven not to have a unique solution. The factorization is a time consuming and computationally expensive operation, needing to be recomputed every time a new rating is entered.

On the other hand, the complex model produced by this technique can be prepared offline, and made ready for almost instantaneous recommendations to be made. The computation required to produce the model, however, is several orders of magnitude higher than the one needed when using a Memory Based Collaborative Filtering, in which the penalty is paid online.

Among Collaborative Filtering Recommender System, an alternative approach to the typical User Based strategy, is to look for similar items instead of similar users. This approach was first taken by Amazon.com which holds a patent to the technology from 2003 [36]. This algorithm is termed Item Based [52, 18, 48, 26], and its major advantage is that products change much slower than users' interaction with the system, and therefore similarities computed hold true for longer, allowing some form of cache to be used.

The second broad category of Recommender Systems that has widespread use, is termed Content Based [43, 48, 26]. Under this paradigm the recommender is built to look for items that are similar to those that the user has purchased or liked in the past. The tool of favour in this domain is the TF-IDF, or Term Frequency – Inverse Document Frequency, which, defined loosely, is a measure of how important a term is inside a collection of documents.

For example, if we had five different books, one being "El Quijote de la Mancha", the term Quijote would be highly discriminative, because it will likely only appear in one of the five books. In other words, in order to identify this book, looking for the word Quijote would suffice. However, if all the books discussed the Quijote, then the term would stop being discriminative and other terms would need to be used instead.

The technique of TF-IDF is used in Content Based Recommender Systems as a way to identify important trends in the user's taste. For example, if a user likes English drama books, then a TF-IDF analysis would isolate the relevant entries reflecting these specifications from the database. In general, the user's profile is assembled as a weighted vector of item features, where the weight represents the relevance of a given feature for a given user.

Pandora.com is one good example of a Content Based Recommender System, as it catalogues songs by features, and then strives to present the listener with melodies that are similar to those previously liked. It does so, not by matching a user profile to neighbour candidates, but by searching features within songs.

One important drawback of a Content Based Recommender System is that it tends to recommend the same type of item to the user, over and over. In order to be able to recommend a different type of item, the user would have to have rated or show interest in the new type of item. This is a problem that a Collaborative Filtering algorithm does not have, since the match here is done between neighbouring users with similar tastes, but different items rated.

This particular limitation of the Content Based strategy gave rise to a Hybrid class of Recommender Systems [26, 11, 12], which were researched as a way to try to get the best from both worlds. In these algorithms, recommendations are produced by a weighted function of both strategies, or by feeding the second strategy, the result from the first.

In this thesis, we concentrate on Memory Based Collaborative Filtering Recommender Systems, and propose ideas and algorithms to improve the similarity search, and thus the quality of the output recommendation. It is believed that there is still plenty of space to improve Memory Based approaches, to compete with their heavy weight Model Based Collaborative Filtering counterparts.

1.2 Objectives of this Work

Recommender Systems have, without a doubt proven themselves an asset technology in the information age. They not only provide a tailored way to focus information to the needs of a particular user, but do this seemingly, without the need for human intervention. Since their inception, they have been an important focus of research in the discipline of Data Mining and Machine Learning, which for the past twenty years has been searching for ways to improve the reliability of Information Retrieval Systems at reflecting a user's taste embedded in a target profile for Information Filtering Systems to produce more meaningful and relevant ways to present yet unseen information to a target user.

While the last decade has seen a concentrated effort in exploring Model Based approaches via Matrix Factorization techniques, Memory Based approaches still remain the principal technique employed in commercial implementations, and it is strongly believed that there is ample space to improve this technology further. For this reason, we have chosen to concentrate on learning and improving Memory Based Collaborative Filtering Recommender Systems.

Clearly, as stated in the introduction in which the Netflix.com million dollar prize competition was described, improving the error metric of a robust Recommender System is not an easy task. In this particular competition, it took the winning team three years to achieve this milestone, and it made use of over 100 different strategies combined to defeat "Cinematch", Netflix.com's own recommender system [31]. But the prize did teach a valuable lesson: there are still more than a handful of simple strategies that can be used to better the recommendation results.

In this work we set two major objectives: (1) to fully understand and correctly employ User and Item Memory Based Collaborative Filtering Recommender System, and (2) to search for strategies aimed at improving the performance of the system. Four approaches of varying complexity resulted from our research. They have been developed and tested in this Thesis, and will be reported in the following sections.

1.3 Summary

In this section we have presented a brief history of the evolution of Recommender Systems, stating their raison d'être and their commercial relevance. Since their inception into commercial products, recommender systems have matured to be ubiquitous in front-end websites where products are offered to shoppers. The whole of the online shopping experience would be nowhere near what it is today without their contribution, to match people's taste to back-end catalogues.

The different types of Recommender Systems currently developed have been presented, namely the Collaborative Filtering and the Content Based approach, and within the former one, the two Memory Based types have been introduced, namely the User Based and the Item Based. The rest of this thesis will focus on the Memory Based Collaborative Filtering formulation.

Lastly, the objectives sought on this work have been stated: learn to use Memory Based Collaborative Filtering Recommender System, test it, and offer novel ways to improve its performance as measure by the MAE and the RMSE. The following sections of this thesis will present four different approaches developed and tested in an attempt to achieve this objective.

1.4 Reader's Guide

The following sections of this thesis are arranged as follows:

Chapter (2) will present the State of the Art in Memory Based Collaborative Filtering algorithms. Relevant formulations will be enumerated, and their advantages and disadvantages discussed. Chapter (3) will present four different algorithms that have been developed and tested as a means to improve the performance of the type of Recommender System studied. Chapter (4) will present our results when testing basic formulations of User Based and Item Based algorithms and our four variations, when applied to the well known MovieLens dataset. Description of the experiments, results and statistical analysis will be included therein. Lastly, Chapter (5) will present a discussion of the results, our conclusions, and some ideas for future work.

Chapter 2

State of the Art

2.1 Historical Overview

Collaborative Filtering algorithms can be traced back to the Tapestry project developed by Xerox PARC in 1992 [20]. In fact, the term Collaborative Filtering itself was coined during this project. Tapestry was a platform that allowed users to tag emails with annotations, and then provide a means to search the emails by using queries. The system was not automatic, but provided a step forward in filtering information by means of a criteria and a user's intent.

The first automatic Collaborative Filtering system was GroupLens from 1994, a product of the GroupLens research group [29, 47]. Their platform provided personalized rating predictions of UseNet news articles using a user neighbourhood strategy combined with a Pearson Correlation similarity function to derive a weighting, and a weighted average of the neighbours to yield the final prediction. Of this research group is the MovieLens dataset, routinely used when assessing Recommender Systems, and employed in the analysis part of this thesis.

Following GroupLens, was the Ringo music recommender from 1995 [55] and the Bellcore video recommender from the same year [24]. Ringo used a similar strategy than GroupLens, but constrained the Pearson Correlation similarity function. They also limited inclusion of neighbours by setting a threshold, instead of including all neighbours as done in GroupLens. Bellcore took a similar approach at selecting neighbours but computed the prediction using linear regression.

The first comparative analysis of neighbourhood based Collaborative Filtering algorithms was done in [10] in 1998. In that paper, the authors contrasted results obtained by using Pearson Correlation and Cosine Vector similarity, finding the first to work better than the latter.

Recommender Systems received a big push forward with the adoption of the technology by Amazon.com at the end of the 1990's. Their own implementation is based on the similarities of items rather than users, and in described in [36]. This allows the company to make claims such as: "users who bought this item also bought these other items".

Another significant move forward occurred during the Netflix.com million dollar prize from 2006. The challenge there was to improve "Cinematch", the proprietary Recommender System of Netflix.com by 10% or more when measured by the Root Mean Square Error (RMSE). The prize was only awarded three years later to a team resulting from the collaboration of some of the top contenders. An important contribution of the competition was the introduction of Model Based approaches to the problem of Collaborative Filtering.

2.2 Recommender Systems

Two families of Recommender Systems exist, namely the Content Based and the Collaborative Filtering approaches. In this Thesis we study solely Collaborative Filtering algorithms, which, due to the nature of the data that they processes (a sparse matrix of Ratings, from Users, about Items – a User-Item matrix), it can take the form of User Based or Item Based depending on the perspective. This distinction will be presented in Section (2.2.1). The Collaborative Filtering algorithm can be formulated as Memory Based or Model Based. We only study the Memory Based formulation, but present the differences between both in Section (2.2.2).

The Collaborative Filtering algorithm sits on top of an assumption which appears to hold valid to a large extend: "there are clusters of users who behave in a similar way and have comparable needs and preferences" [26]. This assumption means that if two people like a given item, and rate it similarly, they will likely share the enjoyment of many other items that can potentially be recommended.

If one of two similar users enjoyed a particular item that the other user is not aware of, this premise says that this item can confidently be recommended to the user, because similar users tend to like and rate similarly. This observation, however simple, is very powerful, as shown by the huge success of Recommender Systems.

In contrast to Collaborative Filtering, Content Based Filtering uses a different approach to recommendations. Under this paradigm, ratings are unimportant, instead, what is sought is to identify similar content. For example, if a user has shown interest in, say, a given book, the Recommender System will analyse the content of that book with the aim of finding other similar books it can offer. Algorithms to match similar content generally make use of counting features and establishing frequencies, in order to come up with high discriminant keywords.

One major drawback of the Content Based Filtering algorithm is that it tends to over-specialize the search. Once a user shows interest in a particular item, the Recommender System will work hard to find other items with similar content it can offer. It will not find unrelated items to the one singled out, a feature called "Serendipity" [44], and deemed one of the strongest assets of a Collaborative Filtering.

In a Collaborative Filtering algorithm, the match is done at the user or the item level, however not with respect to content but with respect to ratings. That is, if two users show interest in similar disconnected items, say, a particular book, a particular CD, and a particular video, if one of these users also showed interest in a given news article, this article could be recommended to the other user.

Clearly, with disjoint item classes the assumption is stretched well beyond what was intended, but the point is made: the second user will be presented with a completely unexpected item, which, if liked, will give rise to "Serendipity" or lucky coincidence, effectively broadening the scope of the recommendation.

Schafer et. al. [52] summarize clearly some of the key questions in which a Collaborative Filtering algorithm tries to assist. They are: *Help me find new items I might like*, *Advise me on a particular item*, *Help me find a user (or some users) I might like*, *Help our group find something new that we might like*, *Help me find a mixture of "new" and "old" items*, *Help me with tasks that are specific to this domain.*

To be successful at this task, a Collaborative Filtering algorithm has to tackle two issues: identify similar users or items to the target one, and average their (potentially weighted) ratings, in order to predict unrated items, so that the ones with the highest score can be recommended to the user.

2.2.1 User vs Item Based

Two different types of Memory Based Collaborative Filtering algorithms exist, namely User Based and Item Based, depending on which dimension of the user-item rating matrix is used to find similarities. Each one of these two strategies has pros and cons. User Based algorithms will search for "like minded individuals" following the assumption that similar users like similar items. Item Based algorithms will search for "item rated similarly by various users" following the assumption that if many users rated two items similarly, they will likely be similar items and worthy of recommendation; again, similar items are generally liked by similar users.

This second strategy also termed Item-Item Recommendation, was first proposed in [51] and thence developed by Amazon.com [36] as part of their in-house Recommender System. The aim of the Item Based approach is to fix what is believed to be an important drawback of the User Based algorithm in real practice: it does not scale well.

Both methods need to identify the N-Nearest Neighbours to the target sample, be this one a user or an item. However, databases tend to have many more users than items, and so the search is much harder in the dimension of the users. In other words, in real life applications, users tend to interact (and therefore, change) at a much higher rate than items do; the latter ones tending to be rather fixed for a given online shop or community [18].

The scalability drawback of a User Based approach is not always the central issue at the time of choosing one algorithm over the other one. In fact, the decision has much more to do with the nature of the domain. For example, [44] argues that in the context of news, the item dimension changes much faster than the user base, and so the Recommender System should favour a User Based approach.

The author also makes an important point when stating and Item Based algorithm can be helpful in offering an explanation as to why an item was recommended. Arguing that similar users have purchased an item is less compelling than arguing that a given item is recommended because it is similar to other items purchased in the past.

2.2.2 Memory vs Model Based

Collaborative Filtering algorithms that compute similarities among neighbours in-memory, are appropriately termed Memory Based. Here, the database of entries is loaded into memory and used directly to generate a recommendation. In contrast to this strategy, there is the Model Based approach, in which a model is derived and used to compute the recommendation. Generally, the Model Based approach produces a model offline, while the recommendation takes place online, and is virtually instantaneous.

Model Based algorithms received a significant push in the research community after the million dollar prize competition opened by Netflix.com in 2006. During the three years that took to win the competition, more and more contenders adopted Model Based approaches as part of their strategy. At the end, the winning team claimed to have used a combination of over 100 different strategies composed into a weighted ensemble, where the principal algorithms employed were Model Based Matrix Factorization techniques [31].

In Matrix Factorization, the users and items ratings matrix is factorized into two matrices, which are said to reveal latent or hidden factors of the rating [26]. These factors, for example, could be the genre of a movie, as in drama or comedy, but oftentimes the factors have no obvious correlation to reality. Nevertheless, Matrix Factorization algorithms have today the upper hand on Recommender Systems as their performance is superior to that obtained by Memory Based approaches.

The obvious drawback of the Model Based method is that the model, not only takes substantially longer to compute, but needs to be computed anew if the matrix of data changes, which happens every time a new user enters a rating. Generally, small changes are left unprocessed, but when they become substantial, the model needs to be re-trained. Memory Based approaches do not suffer from this problem because the similarity is computed every single time a recommendations is sought.

2.3 User Ratings

Collaborative Filtering Recommender Systems rely on ratings to provide recommendations. These preferences can take two forms [18], they can either be explicit, as in the case of a user assigning a valuation to an item, or implicit, as in the case where the system concludes that users either like or dislike a product by studying their actions, such as looking at purchases or screen clicks.

In the context of Collaborative Filtering, ratings are a numerical value representing how much a user likes or dislikes a given item, and are the basis for similarity measures used to find like minded candidates. In this section we review how ratings are gathered, the numerical scales in general use, and how to compensate for differences in rating style among users.

2.3.1 Explicit vs Implicit

An explicit rating, occurring when the user gives direct appraisal to an item, obviously carries an immense value, since the system is actually being told what the user feels about the item in question. However, human psychology is complex, and the inference of taste from a ranting is not as consistent as one would hope [44]. A user might think highly of a given item, but for some unknown reason decide to give a low rating instead. Social pressure. Viral trends. Cool factor. All these external influences affect the way users rate, and impact the recommendations of the system.

Moreover, the rating scale, even though it is usually standardized, can be perceived differently by different voters. Two people liking a film a lot, might rate it completely different, only because one is more generous in the marking, while the second one consistently rates lower to avoid inflation.

Another problem with the explicit rating mechanism is one of memory. Generally, users appraise an item close to the moment when the item was inspected; a book just read or a movie now watched. These items are much more fresh in memory, compared to other items reviewed in the past. This makes it difficult for a user to be objective, and to relativise the rating to the scale used in the other instances.

As a result, a rating of “excellent” for a particular film watched a month ago could very well not stand up to the competition of a new film watched now, but be equally rated “excellent”. And since users tend to not change an old rating, both items would now hold the same level of appreciation.

But even if a user does remember an old item well, taste changes through time [44]. And if ratings are not adjusted to reflect the new preferences, the old ones will continue to affect any recommendation of the system, rendering it less effective.

Implicit ratings try to minimize the impact of human psychology by not involving the user in the derivation of the appraisal. Preferences inferred are less biased since they are not affected by what the user thinks he should rate [44]. However, an inferred system is not devoid of problems. One common strategy to guess at the user’s taste is to infer it by studying how much time he stayed browsing a particular item; but, how is an algorithm to know if the user is truly studying an item with agreement or disagreement? Or worse yet, what if he left the page open while going to prepare a cup of coffee?

View pages and items purchased have also been used as a means to infer taste [18]. The rationale is that an item purchased is an item liked, and a page visited is a page of interest. Naturally, one could buy an item for a friend, and end up in a page by mere clicking a link by mistake. Also, in some cases, more than one person browses from the same account, creating effectively a potpourri of tastes, with little targeting value.

The main advantage of implicit ratings over explicit ones is that the user is not troubled by having to answer questions in order to train the algorithm. However, both strategies can easily be combined to enhance the knowledge of a particular user’s taste.

2.3.2 Rating Scales

In contrast with a Content Based Recommender System in which the content of the item is mined for similarities against the content of other items, in Collaborative Filtering, the similarities sought are the subjective preferences of users, and thus require some form of metric to be used, in order to equate them. In general, they take one of three forms [52]:

- Unary: Good or Purchased
- Binary: Like/Dislike
- Integer: 1-to-N (e.g. 1-5 stars)

The Unary scale is used to signal that a user has shown interest in a particular item. It can be implicit as in the case of a view page or a purchase, or explicit by adding an appropriate button, like the Google “+1” or the Facebook “Like”. It tells nothing about the items the user does not like, but it is non-intrusive, and in many cases, enough to learn about a user.

The Unary scale is typically inferred by the system, by looking at the user’s actions, instead of asking for feedback. While the user’s is not expressively stating a preference, his preference is still learned by way of the choices he makes.

The Binary scale is explicit, and takes the form of two buttons, one conveying a positive feeling, and one conveying a negative one. An example of this scale is found in YouTube.com where users can click on a “thumbs up” or “thumbs down” icon. They confer general feelings about an item without much gamut, however they are good polarizers with minimal intrusiveness.

Lastly, the Integer scale is akin to an hotel stars or a restaurant forks rating, and is also explicit. It gives the largest spectrum to express taste, as it lets the user express himself with more than a general direction. This last rating is customarily found when rating films or books.

2.3.3 Normalizations

One of the problems faced with using ratings as a means of representing taste, is that each user has personal interpretation of the scale. While one rater might tend to give high marks to films he likes, another rater might keep the highest grades for exceptional movies. The question then is how similar both users are, and how to user their ratings appropriately.

There are two widely used systems to compensate for variations in the rating approach by different users, one is called mean-centering, and the other is called Z-score [48].

The mean-centering algorithm re-maps a user’s rating by subtracting the mean value of all his ratings, effectively signalling if the particular rating is positive or negative when compared to the mean. Positive values represent above-average ratings, negative results represent below-average ratings, and zero represents an average rating. The formulation is given by [1]:

$$h(r_{ui}) = r_{ui} - \bar{r}_u \tag{2.1}$$

where $h(r)_{ui}$ represents the mean-centered rating of user u for item i, r_{ui} represents the actual rating of user u for item i, and $\bar{r}_u$ represents the mean rating of user u across all items rated.

When adopting the Item Based approach, the equation simply becomes:

$$h(r_{ui}) = r_{ui} - \bar{r}_i \tag{2.2}$$

where $h(r)_i$ represents the mean rating of item i across all users that rated the item.

The Z-score approach is very similar to the mean-centering, but it is normalized by the standard deviation σ_u or σ_i depending on whether we apply the User or the Item based approach [1]:

$$h(r_{ui}) = \frac{r_{ui} - \bar{r}_u}{\sigma_u} \tag{2.3}$$

Normalization of ratings does come with a word of caution. If a user only gave high ratings to all items reviewed, the mean-centered approach would yield them average, and any rating below the highest would be a negative one, even if it is fairly high in the scale. Likewise, if a user has rated all items with the same exact grade, the standard deviation will be zero, and the Z-score won't be computable [1].

2.4 Similarity Metrics

At the heart of the Collaborative Filtering algorithm lays the strong assumption that "similar users like similar items". While this assumption appears to hold true to a great extent, the key question becomes: how do we define and identify similar users in order to find similar candidate items we can recommend? Any such formulation needs to consider the data we have available, a sparse matrix of User-Item Preferences, where many users have likely only rated a small proportion of many the items in stock.

In the GroupLens Recommender System, the first automatic such system, the Pearson Correlation was suggested as a suitable measure of user similarity [29]. Various similarities have been suggested, namely the Euclidean Distance and the Cosine Distance, among others. Which one works best in Collaborative Filtering Recommender Systems is openly argued, having some authors claim that the Pearson Correlation is the best suited algorithm for User Based approaches [10, 23], while identifying the Cosine Distance on Item Based approaches [26].

Whichever similarity function a Recommender System uses, it is important to keep in mind that each user rates items in their own way (as seen in the previous section), and that similarity ratings of common popular items might not necessarily reflect the similarity of controversial items which at the end might have more value in a recommendation [26].

2.4.1 Pearson Correlation

The Pearson Correlation similarity function is perhaps the best known algorithm for User Based Recommender Systems. It was first introduced by the GroupLens project in 1994, and used ever since as baseline for comparisons.

When using the Pearson Correlation, the similarity is represented by a scale of -1 to +1 where a positive high value suggests a high correlation, a negative high value suggest inversely high correlation (when one say True the other says False), and lastly a zero correlation indicates uncorrelated samples.

The User Based Pearson Correlation similarity equation is defined as:

$$s(u,v) = \frac{\sum\limits_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum\limits_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_u)^2 \sum\limits_{i \in \mathcal{I}_{uv}} (r_{vi} - \bar{r}_v)^2}} \tag{2.4}$$

where $s(u,v)$ represents the similarity of users u and v, $\mathcal{I}_{uv}$ represents the set of items rated by both users u and v, r_{ui} and r_{vi} represent the rating of user u or v for item i, and $\bar{r}_u$ and $\bar{r}_v$ represent the mean rating of user u or v, across all items rated.

In the case of Item Based Recommender Systems, the formulation becomes:

$$s(i,j) = \frac{\sum\limits_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_i)(r_{ui} - \bar{r}_j)}{\sqrt{\sum\limits_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_i)^2 \sum\limits_{u \in \mathcal{U}_{ij}} (r_{ui} - \bar{r}_j)^2}} \tag{2.5}$$

where $s(i,j)$ represents the similarity of items i and j, $\mathcal{U}_{ij}$ represents the set of common users who rated items i and j, and $\bar{r}_i$ and $\bar{r}_j$ represent the mean rating of item i or j, across all users that rated the item.

2.4.2 Cosine Distance

The Cosine Distance similarity function looks at two samples to compare, and studies the cosine of the angle between them in order to quantify the similarity. The scale used in this case is the same as the one used in Pearson Correlation, where +1 and -1 represent high (direct and inverse) correlation, and zero represent no correlation.

In vector form, the Cosine Distance is defined as:

$$cos(\vec{a},\vec{b}) = \frac{\vec{a}.\vec{b}}{\|\vec{a}\|^2 * \|\vec{b}\|^2} \tag{2.6}$$

The User Based Cosine similarity equation, defined as a summation, becomes:

$$s(u,v) = \frac{\sum\limits_{i \in \mathcal{I}_{uv}} r_{ui} r_{vi}}{\sqrt{\sum\limits_{i \in \mathcal{I}_u} r_{ui}^2 \sum\limits_{i \in \mathcal{I}_v} r_{vi}^2}} \tag{2.7}$$

where $s(u,v)$ represents the similarity of users u and v, across all items commonly rated.

In the case of Item Based Recommender Systems, the formulation becomes:

$$s(i,j) = \frac{\sum\limits_{u \in \mathcal{U}_{ij}} r_{iu} r_{ju}}{\sqrt{\sum\limits_{u \in \mathcal{U}_i} r_{iu}^2 \sum\limits_{u \in \mathcal{U}_j} r_{ju}^2}} \tag{2.8}$$

where $s(i,j)$ represents the similarity of items i and j, across all users that rated such items.

2.4.3 Adjusted Cosine Distance

The Adjusted Cosine Distance was proposed as a way to compensate for the fact that users rate items differently. It does so by subtracting the average user ranting from each individual rating. In this case, the formulation on User Based approaches, becomes:

$$s(u,v) = \frac{\sum\limits_{i \in \mathcal{I}_{uv}} (r_{ui} - \bar{r}_i)(r_{vi} - \bar{r}_i)}{\sqrt{\sum\limits_{i \in \mathcal{I}_u} (r_{ui} - \bar{r}_i)^2 \sum\limits_{i \in \mathcal{I}_v} (r_{vi} - \bar{r}_i)^2}} \tag{2.9}$$

where $s(u,v)$ represents the similarity of users u and v, across all items commonly rated.

In the case of Item Based Recommender Systems, the formulation becomes:

$$s(i,j) = \frac{\sum\limits_{u \in \mathcal{U}_{ij}} (r_{iu} - \bar{r}_u)(r_{ju} - \bar{r}_u)}{\sqrt{\sum\limits_{u \in \mathcal{U}_i} (r_{iu} - \bar{r}_u)^2 \sum\limits_{u \in \mathcal{U}_j} (r_{ju} - \bar{r}_u)^2}} \tag{2.10}$$

where $s(i,j)$ represents the similarity of items i and j, across all users that rated such items.

2.4.4 Mean Squared Distance

The Mean Squared Distance similarity is defined as:

$$s(u,v) = \frac{\sum\limits_{i \in \mathcal{I}_{uv}} (r_{ui} - r_{vi})^2}{|\mathcal{I}_{uv}|} \tag{2.11}$$

where $s(u,v)$ represents the similarity of users u and v, across all items commonly rated.

In the case of Item Based Recommender Systems, the formulation becomes:

$$s(i,j) = \frac{\sum\limits_{u \in \mathcal{U}_{ij}} (r_{iu} - r_{ju})^2}{|\mathcal{U}_{ij}|} \tag{2.12}$$

where $s(i,j)$ represents the similarity of items i and j, across all users that rated such items.

Some texts [48] suggest using the inverse of the Mean Squared Distance, as in:

$$\hat{s}(a,b) = \frac{1}{s(a,b)} \tag{2.13}$$

where $\hat{s}(a,b)$ represents the Mean Squared Similarity, and $s(a,b)$ represents the Mean Squared Distance.

2.4.5 Euclidean Distance

The Euclidean Distance similarity is defined as:

$$s(u,v) = \sqrt{\frac{\sum\limits_{i \in \mathcal{I}_{uv}} (r_{ui} - r_{vi})^2}{|\mathcal{I}_{uv}|}} \tag{2.14}$$

where $s(u,v)$ represents the similarity of users u and v, across all items commonly rated.

In the case of Item Based Recommender Systems, the formulation becomes:

$$s(i,j) = \sqrt{\frac{\sum\limits_{u \in \mathcal{U}_{ij}} (r_{iu} - r_{ju})^2}{|\mathcal{U}_{ij}|}} \tag{2.15}$$

where $s(i, j)$ represents the similarity of items i and j, across all users that rated such items.

This equation can readily be seen to be the Mean Squared Distance with an added square root.

In some Recommender System implementations [6], the similarity is defined as:

$$\hat{s}(a, b) = \frac{1}{1 + s(a, b)} \tag{2.16}$$

where $\hat{s}(a, b)$ represents the Euclidean Similarity, and $s(a, b)$ represents the Euclidean Distance.

This has the effect of limiting the rating to the range of (0..1].

2.4.6 Spearman Correlation

The Spearman Correlation is an alternative formulation to the Pearson Correlation, but instead of using the actual rating values, it uses a rank of those ratings. If we denote k_{ui} the rank of the rating of item i of user u, then the equation on User Based systems becomes:

$$s(u, v) = \frac{\sum\limits_{i \in \mathcal{I}_{uv}} (k_{ui} - \bar{k}_u)(k_{vi} - \bar{k}_v)}{\sqrt{\sum\limits_{i \in \mathcal{I}_{uv}} (k_{ui} - \bar{k}_u)^2 \sum\limits_{i \in \mathcal{I}_{uv}} (k_{vi} - \bar{k}_v)^2}} \tag{2.17}$$

where $s(u, v)$ represents the similarity of users u and v, across all items commonly rated.

In the case of Item Based Recommender Systems, the formulation becomes:

$$s(i, j) = \frac{\sum\limits_{u \in \mathcal{U}_{ij}} (k_{ui} - \bar{k}_i)(k_{ui} - \bar{k}_j)}{\sqrt{\sum\limits_{u \in \mathcal{U}_{ij}} (k_{ui} - \bar{k}_i)^2 \sum\limits_{u \in \mathcal{U}_{ij}} (k_{ui} - \bar{k}_j)^2}} \tag{2.18}$$

where $s(i, j)$ represents the similarity of items i and j, across all users that rated such items.

The benefit of the Spearman Correlation over the Pearson Correlation is that it avoids the need to normalize the ratings [48].

2.5 Neighbourhood

In theory, the similarity values of all the candidate users can be used at the time of recommending an item to a target user, however, this is not only an impractical approach with large user databases, but including users with low correlation to the target actually hinders the recommendation as it will increase its error [23].

To avoid the inclusion of users uncorrelated to the target, a neighbourhood selection step is generally included in the overall algorithm, in order to filter out those unwanted candidates. Two main methods are commented in the literature, namely, selecting the Top N-Neighbours, and using a Threshold Filtering, which are described in Sections (2.5.1) and (2.5.2).

2.5.1 Top N-Neighbours

In the Top N-Neighbours approach, the best N correlated candidates are filtered in, while the others are filtered out, and not used in subsequent steps of the recommendation process. The larger the set of included users, the more, the variance in their ratings will be averaged out, but the overall quality of the prediction decreases. This also results in a loss of "serendipity", which was argued to be a good feature of Collaborative Filtering, as it gives rise to unexpected good recommendations [23].

The ideal number of users to include, is a parameter that needs to be set by cross-validation since it is highly dependent on the dataset. One important outcome of using the Top N-Neighbours is that the Recommender System can later support the recommendation by providing proof via the list of neighbours [1]. Everyday commercial use of Recommender Systems has taught that supporting the recommendation is very important as it explains the user why a particular item was recommended, and dissipates the feeling of being offered something completely at random.

Employing Top N-Neighbours also assist in the scalability of the whole system since using every user available in a large database of users would result in a significant performance hit. Of note is that when the number of users largely exceed the number of items, an Item Based approach as discussed earlier in Section (2.2.1), might be a good strategy to adopt.

One problem with this technique is that a set number of neighbours will not likely satisfy every user in the system. Some users might be easier to correlate to others, while some other users might require more neighbours in order to compute good predictions, because they are poorly correlated to the majority of users in the database. Also, in some cases, including more users will only add noise because these poorly correlated candidates will effectively lower the recommendation quality.

2.5.2 Threshold Filtering

The Threshold Filtering approach looks to meet a minimum satisfactory level of similarity for a neighbour to be included in the output set. This strategy fixes the problem of using a single neighbourhood size for all candidates, but is not devoid of its own problems.

Setting too high a threshold will result in very good correlated neighbours, but for some users that are not easily correlated to those in the database, the result might be too few neighbours, with very low quality recommendations. Setting a lower threshold will increment the number of neighbours accepted, which defeats the purpose of this strategy [22].

In general, as the threshold is set tighter, the overall recommendation error drops, but less users can reliably be recommended items. As the threshold is loosen, the recommendation error increases, but more users can be recommended items. As always, there is a trade off that needs to be carefully considered for each Recommender Systems in question.

2.6 Rating Prediction

Once similarities for the chosen neighbourhood are computed, the last step of the recommendation process as defined in Section (2.6.1) involves deriving an estimation of the unknown rating for the items that the user's neighbours have rated, but that the target user has not. With a list of rating estimates for unrated items, the Recommender System will simply sort by decreasing value, and recommend the Top I-Items of the list to the target user.

In order to produce an estimation of a rating for an unrated item, a weighted average of all available ratings is done, using as weights the correlation values computed with the similarity functions. The implicit assumption here is that all users rate out of the same statistical distribution [22]. The equation to compute a rating estimate in a User Based system is given as follows:

$$\hat{r}_{ui} = \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv} r_{vi}}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|} \tag{2.19}$$

where $\mathcal{N}_i(u)$ represents the number of neighbours that have item i in common with user u, of which v is a particular neighbour user, and w_{uv} is the similarity between the user u and one of its neighbours v.

For an Item Based system, the corresponding equation would be:

$$\hat{r}_{ui} = \frac{\sum\limits_{j \in \mathcal{N}_u(i)} w_{ij} r_{ju}}{\sum\limits_{v \in \mathcal{N}_u(i)} |w_{ij}|} \tag{2.20}$$

where w_{ij} is the similarity between the item i and one of its neighbours j.

The Rating Prediction can take into consideration the Rating Normalizations discussed earlier in section (2.3.3). When using the Mean Centering approach, equation (2.19) becomes:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv}(r_{vi} - \bar{r}_v)}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|} \tag{2.21}$$

Likewise, for Item Based systems, equation (2.20) becomes:

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum\limits_{j \in \mathcal{N}_u(i)} w_{ij}(r_{ju} - \bar{r}_j)}{\sum\limits_{j \in \mathcal{N}_u(i)} |w_{ij}|} \tag{2.22}$$

If instead of using the Mean Centering approach we were to use the Z-score approach, equation (2.19) would instead be stated as:

$$\hat{r}_{ui} = \bar{r}_u + \sigma_u \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv}(r_{vi} - \bar{r}_v)/\sigma_u}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|} \tag{2.23}$$

And for Item Based systems, equation (2.20) becomes:

$$\hat{r}_{ui} = \bar{r}_i + \sigma_i \frac{\sum\limits_{j \in \mathcal{N}_u(i)} w_{ij}(r_{ju} - \bar{r}_j)/\sigma_i}{\sum\limits_{j \in \mathcal{N}_u(i)} |w_{ij}|} \tag{2.24}$$

In [1], the author states that Z-score normalization, despite having the added benefit of taking into consideration the variance in the rating, it is more sensitive than Mean Centering and oftentimes it predicts values outside the rating scale.

2.6.1 Recommender Algorithm

A pseudo-code description of the overall Recommender Algorithm can be included for clarity. It is comprised of the following steps [1]:

Standard Top N-Neighbours User Based Collaborative Filtering Algorithm

```
1  **Input**: User-Item Rating matrix $R$
2  **Output**: Recommendation lists of size $l$
3  **Const** $l$: Number of items to recommended to user $u$
4  **Const** $v$: Maximum number of users in $\mathcal{N}(u)$, the neighbours of user $u$
5  **For** each user $u$ **Do**
6      **Set** $\mathcal{N}(u)$ to the $v$ users most similar to user $u$
7      **For** each item $i$ that user $u$ has not rated **Do**
8          Combine ratings given to item $i$ by neighbours $\mathcal{N}_i(u)$
9      **End**
10     Recommend to user $u$ the top-$l$ items with the highest predicted rating $\hat{r}_{ui}$
11 **End**
```

The algorithm takes as input (Line 1) the User-Item sparse matrix of Ratings R and outputs (Line 2) a recommendation list of size l (Line 3), which is set according to the needs of the system. The parameter k (Line 4), which is generally set by cross-validation, dictates the size of the neighbourhood of users to use, to find similar candidates to the target.

The bulk of the algorithm loops through the list of users (Line 5), and for each one of them it finds the most similar k users to this target (Line 6). The similar users are obtained by using a similarity function as described in Section (2.4), being the Euclidean Distance, the Pearson Correlation, and the Cosine Distance the most used ones. The selection of the most similar users is done by using a neighbourhood selection algorithm as described in Section (2.5), being the Top N-Neighbours or the Threshold Filtering algorithms the usual choices.

Then, for each item on stock that the target user has not rated (Line 7) we combine the ratings given by the candidate neighbours (Line 8) to produce a single estimate of each missing rating. Finally (Line 10), we recommend the target user the top-l items that received the highest rating (Lines 7-9), assuming that the items not rated by the user are items that the user is not aware of.

2.7 Assessment Metrics

In this section we present the usual metrics used to assess the performance of Recommender Systems. The first metric is Coverage, which represents the percentage of unrated items by given users that the algorithm can estimate. Because of a lack of similar users to a particular target user, or too few ratings collected for the target user, not all unrated items will be estimated.

The second metric is Accuracy, and represents the error encountered when estimating an unrated item of a given target user. When assessing a Recommender System it is customary to perform a leave-one-out cross validation, where the known rating is left out of the User-Item Ratings matrix, and estimated by the algorithm. Doing this for each and every known rating would yield the overall Accuracy of the system.

2.7.1 Coverage

Coverage is a measure of the percentage of the estimates that the Recommender System algorithm was able to complete. Generally, because of choice of Neighbourhood, Coverage is not 100%, but a lower value.

As stated in section (2.5) the selection of the type of Neighbourhood strategy to use has an important impact in the number of ratings that the algorithm can produce. For example, when using the Threshold Filtering technique, the higher the threshold, the more similar the samples, but the smaller the neighbourhood, which in turn results in some number of samples not finding similar candidates, and thus their unrated items not being estimated.

A high coverage means that the Recommender System is able to recommend items to many users, regardless of their peculiarities (some users are hard to match). This, naturally, comes at a cost, which is measured by the performance of the algorithm. The larger the neighbourhood, the more outliers are accepted, and the worse the recommendations become.

2.7.2 Accuracy

Accuracy is a measure of the performance of the algorithm. It quantifies how the estimations produced by the strategy deviate from their known values. The smaller the error, the better the Recommender System works. Accuracy is closely related to Coverage in the sense that once one is high, the other one tends to be low. A compromise must be found between these two parameters for each Recommender System to work at it best.

Two types of Accuracy measures are in wide spread use in the literature; they are the Mean Absolute Error (MAE), and the Root Mean Squared Error (RMSE). Their equations follow:

$$MAE(r, \hat{r}) = \frac{\sum\limits_{a \in \mathcal{N}_{ui}} |r_a - \hat{r}_a|}{\|r\|} \tag{2.25}$$

$$RMSE(r, \hat{r}) = \sqrt{\frac{\sum\limits_{a \in \mathcal{N}_{ui}} (r_a - \hat{r}_a)^2}{\|r\|}} \tag{2.26}$$

MAE and RMSE do present some drawbacks when used to measure the Accuracy of a Recommender System [39]. For example, if a rating is not available, then that ratting is not reflected in the error. Also, in the case of MAE, the error is linear, meaning that the size of the error is the same at the high and the low ends of the similarity. The RMSE does penalize more the larger errors, which might make sense in the context of Recommender Systems.

2.8 Improvement Strategies

Several improvement strategies have been developed for Recommender Systems to supplement the core of the algorithm. One of such strategies is Significance Weighting, which looks to weight the contribution of each candidate user according to how many common items it has rated when compared to the target user. Naturally, some candidate users are more similar to the target than others, and this formulation attempts to reflect this fact.

A second improvement is the Default Voting, which aims to use more samples when computing the similarity than those strictly available. When comparing a candidate user to the target user, the similarity algorithm would only consider ratings present in both, the candidate and the target. The Default Voting computes an average rating, and sets all missing ratings to this value, effectively extending the number of common ratings.

Lastly, we describe the Context Aware strategy which makes user of other information available to compute the similarity. For example, ratings might vary depending on weekday, or prices might change depending on season. Strategies to incorporate these subtleties have been proposed, and are presented in this Section.

All three improvement strategies described here have been studied, and in the case of Default Voting and Context Aware, have been used as basis for new proposals put forward in this Thesis.

2.8.1 Significance Weighting

Both Neighbouring strategies presented in section (2.5), when fine tuned properly, filter out lowly correlated samples from entering the recommendation, and driving the performance of the algorithm lower. However, what the Neighbouring strategies do not address is the fact that not all accepted samples are equally correlated to the target. Some samples share more common items than others, rendering them better at the time of computing the recommendation.

Such a feature proposes to weight the significance of a candidate by the inverse of the number of common items it shares with the target in question [22]. The idea is that despite a high similarity computation, if a candidate and a target only share very few items, its contribution should be weighted down, because they are not broadly similar, only apparently similar. Neighbours with few common samples generally prove to be bad predictors for a target [23].

At the heart of this strategy lays the use of a measure of confidence in the similarity obtained. The more items two samples share in common, the more reliable they are for producing a recommendation, and the more they should weight as part of the overall recommendation. Likewise, poorly shared items should be heavily weighted down so that they don't influence the recommendation that much.

The formulation of this strategy on User Based approaches is expressed as follows:

$$w'(uv) = \frac{|\mathcal{I}_{uv}|}{|\mathcal{I}_{uv}| + \beta} w_{uv} \tag{2.27}$$

where $w'(uv)$ is the new weighting after taking into consideration the number of commonly weighted items, $|\mathcal{I}_{uv}|$ represents the number of co-rated items, and $\beta > 0$ is a parameter obtained by cross-validation.

On Item Based approaches, the equation would become:

$$w'(ij) = \frac{|\mathcal{U}_{ij}|}{|\mathcal{U}_{ij}| + \beta} w_{ij} \tag{2.28}$$

Al alternative formulation of Significance Weighting is found in [37], which, for User Based approaches has the form:

$$w'(uv) = \frac{\min(|\mathcal{I}_{uv}|, \gamma)}{\gamma} w_{uv} \tag{2.29}$$

where $|\mathcal{I}_{uv}|$ represents the number of items that users u and v have co-rated, and $\gamma > 0$ represents a threshold value of co-rated items.

And for Item Based approaches becomes:

$$w'(ij) = \frac{\min(|\mathcal{U}_{ij}|, \gamma)}{\gamma} w_{ij} \tag{2.30}$$

where $|\mathcal{U}_{ij}|$ represents the number of users who rated both items i and j.

2.8.2 Default Voting

An alternative approach to deal with similarities, sharing few items in common is termed Default Voting [34]. In this strategy, a default value is computed for all items missing a rating. This has the effect of augmenting the number of co-rated items, which, as discussed previously, improves the accuracy of the recommendation.

The strategy is easily adapted to the similarity algorithms. For example, in the case of User Based approaches, the Pearson Correlation similarity expressed in equation (2.4), becomes:

$$s^{\rho}(u,v) = \frac{\sum\limits_{i \in \mathcal{I}_{uv}} (r^{\rho}_{ui} - \bar{r}_u)(r^{\rho}_{vi} - \bar{r}_v)}{\sqrt{\sum\limits_{i \in \mathcal{I}_{uv}} (r^{\rho}_{ui} - \bar{r}_u)^2 \sum\limits_{i \in \mathcal{I}_{uv}} (r^{\rho}_{vi} - \bar{r}_v)^2}} \tag{2.31}$$

where $s^{\rho}(u,v)$ represents the similarity of users u and v, across all items commonly rated, with Default Voting enabled.

In the case of Item Based Recommender Systems, equation (2.5) becomes:

$$s^{\rho}(i,j) = \frac{\sum\limits_{u \in \mathcal{U}_{ij}} (r^{\rho}_{ui} - \bar{r}_i)(r^{\rho}_{ui} - \bar{r}_j)}{\sqrt{\sum\limits_{u \in \mathcal{U}_{ij}} (r^{\rho}_{ui} - \bar{r}_i)^2 \sum\limits_{u \in \mathcal{U}_{ij}} (r^{\rho}_{ui} - \bar{r}_j)^2}} \tag{2.32}$$

where $s(i,j)$ represents the similarity of items i and j, across all users that rated such items, with Default Voting enabled.

In both expressions, the term r^{ρ}_{ab} is defined as follows:

$$r^{\rho}_{ab} = \begin{cases} r_{ab} & \text{if } r_{ab} \text{ is defined} \\ \rho & \text{otherwise} \end{cases} \tag{2.33}$$

Alternative expressions for other similarity functions can readily be derived to use Default Voting.

2.8.3 Context Aware

Collaborative Filtering Recommender Systems tend to limit themselves to process the ratings matrix formed by users and items, which is inherently $2D$. However, in many instances, more information is available, that could be used to improve the results of the recommendation. Context Aware Recommender Systems [48] take advantage of the added dimensions using $3D$ or higher order matrices as input to the system.

For example, information about movies might include more than just the title, like the genre, the year of release, the director, the actors, and other. When searching for similar movies, a Context Aware system could filter out particular unwanted directors, while including genres of interest, and thus focus the recommendation better.

Another example of the application of Context Aware Recommender Systems is when recommending items for a particular time, like a weekday, a weekend, or a holiday. In a trip recommender systems, the time dimension could play a prominent role, since a user might rate a tourist attraction as very good on a weekend, but very poor on a weekday. In a flight recommender system, taking into consideration the price changes occurring at various time frames, such as holidays or peak season, would account for a performance increase when recommending flights.

Clustering has also been proposed [42] as a means to compartmentalize the information available. Under this strategy, smaller dimensionality spaces are created, reducing the large search space the algorithm is usually presented with. With the smaller sub-spaces created, a conventional Collaborative Filtering algorithm can be applied on each partition, reducing substantially the computational cost, and sometimes improving the results due to specialization of the different clusters.

2.9 Typical Problems

Properly tuning a Collaborative Filtering Recommender System is not an easy task, but is a fundamental step to ensure the good working order of the recommendation process. Some of the typical problems faced by the system include: *How to recommend items to a completely new user? How to recommend users a new item that has not yet been rated? How to satisfy a non-typical user? How to fight attempts by users to affect the results of recommendations?*

At the end of the day, a successful recommendation is more a function of a good dataset than a good algorithm or strategy. Without reliable data, no algorithm can yield satisfactory results, and by nature, the problem confronted by Collaborative Filtering algorithms is one of Sparseness; users tend to rate only a very small fraction of all the items held in the database.

But clearly, without recommendations the system cannot start functioning, and this problem, termed Cold Start, affects every new deployed Recommender System. How to incite new users to share their preferences with the rest is an important facet of the success of an installation.

Like all things open to the general public, Recommender Systems are not immune to manipulation. If a group of users organise and start rating particular items high, this will affect the whole of the system. Giving positive recommendations to one's own items, while bad ones to competitors is termed Shilling Attacks [58].

2.9.1 Sparsity

One of the hardest problems to deal with at the time of making recommendations is that of sparsity of the information. A user will tend to only rate a few items, and generally the system will hold on file a large number of users with little item ratings recorded for each. In other words, the input matrix of users and items is inherently very sparse.

This has profound effects for the Recommender System. There might be lots of *truly* similar candidate users to a target user, but all similarity algorithms work by matching the ratings that were shared by those users, and without ratings, similar users will not be identified.

A common strategy to deal with the sparsity problem was presented in section (2.8.2) where unknown rating values are replace by averages of the known ratings entered by a user. But this strategy assumes that the missing ratings are efficiently modelled by an average function, when in fact they might well lay low below or high above the known ratings mean.

Sparsity complicates the consolidation of neighbourhoods of similar users, and thus affects the coverage of the algorithm, since some targets will likely not be matched to a sufficient number of candidates from which to derive a recommendation. In fact, the denser the dataset, the better the performance [61].

2.9.2 Cold Start

At the heart of the Collaborative Filtering algorithm lays the user rating. Without ratings, no recommendation can take place for any user of the system. This problem is termed Cold Start, and describes the inability of a Recommender System to start offering recommendations. The problem faced with jump starting these systems is that without recommendations, other users tend to shy away, and will not recommend by themselves, forcing the system to a complete halt.

One way of breaking this pattern is to force new users to rate a number of items before allowing them to use the system. This strategy ensures there are always new ratings coming into the system, while solving also a second problem that affects every newcomer: a new user that has yet to enter his own preferences cannot be recommended anything at all by the system.

While it is possible to offer generalized untargeted recommendations to a new user, it is not a satisfactory solution since the power of the Recommender System is completely waisted. Offering a rating that works for the majority is the exact opposite of seeking novelty and targeting the recommendation to a particular user profile, which is what Recommender Systems excel at.

Lack of new user ratings is not the only problem in a Collaborative Filtering Recommender System. New items added will also lack any ratings, and will thus not be recommended to anybody. This situation is termed Early Rater [44].

Here, the strategy of forcing users to rate the new item could help, however, the problem is generally viewed as being less critical than the new user scenario because novel items tend to be found fast, and ratings follow suit [52]. A good strategy to use with new items is to immediately put the on display so users can readily find and rate them.

2.10 Summary

In this section we have covered much of the technology behind Collaborative Filtering Recommender Systems. We have reviewed the use of Ratings, and presented some strategies employed to compensate for the differences in rating style used by each user. We have then presented the various Similarity Functions in common use on Recommender Systems, which are based on comparing candidates to a target and measuring quantitatively the difference between them. This step was said to be critical to establish a weight representing the similitude between two users. As argued, Collaborative Filtering algorithms are build on the assumption that "similar users like similar items".

The choice of Neighbourhood was discussed, pointing out that the different strategies proposed in the literature attempt to separate the useful candidates from those that would lower the quality of the final recommendation. The process of making a Recommendation was presented next, with equations to incorporate corrections for how users rate items. Coverage, and metrics to assess performance were commented on, and several improvement strategies used to deal with sparsity and unmatched number of co-rated items. Lastly, typical problems encountered in deployed Recommender Systems were listed.

Chapter 3

Proposals

3.1 Description of Proposals

In Section (2.8) we presented three important techniques aimed at improving the accuracy of Collaborative Filtering algorithms. Among the three, Default Voting and Context Aware seemed good candidates to build upon. In this Chapter we will stretch these techniques further in an attempt to extract better results out of them.

The concept of Sparsity, presented in Section (2.9.1) could be singled out as the Achilles' heel of these systems. Sparsity, a result of many users rating only a small portion of many items, hinders the search for similar users or similar items, and hampers the preference estimation of unrated items, which in turn makes it difficult for the Recommender System to output a successful recommendation.

In a perfect system, every user would rate every item, and do so faithfully. Moreover, users would not change their voiced preference over time, and would not be influenced by fashion, trends or other social pressures. But the stark reality is that these shortcomings are present, and commercial implementations do have to deal with them in some way.

The Default Voting strategy was brought up to directly solve the problem of Sparsity in the ratings. Default Voting acts by filling in missing preferences in the matrix of users and items ratings, which generally is only sparsely filled. The values used to fill in the missing preferences of this matrix are generally the average of the item ratings given by each user.

The underlying assumption at play here is that a user would tend to rate all items similarly. However, this would only hold if a user only rated items he either liked or disliked, but not both. However, when assigning the default rating to the unrated items, potentially, those he wouldn't like would become liked, and vice-versa.

A better idea to fill in the sparse matrix is required, and two such strategies will be presented in the following sections. Neighbourhood selection is arguably the most important step of the Recommender Algorithm, since choosing similar users to a target is what enacts our assumption that "similar users like similar items", and therefore supports our subsequent recommendation.

Context Aware is another approach that is believed to be under-utilized. Most commercial Recommender Systems hold much more than user and item preferences. They have item descriptions, user profiles, demographics, statistics; and all this information can be appended to the system as an extra dimension that can be taken advantage of at the time of finding similar candidates.

For example, if one was interested in finding appropriate music to recommend to a user, knowing that he likes rock but hates rap, is a good start, since all CDs involving rock would be filtered in, and all those involving rap would be filtered out, rising the number of potential similar candidates, since the pool now would not only include users liking rock *but* hating rap, but also users liking rock *and* liking rap.

Such a strategy has been researched in this Thesis, and will be presented in the coming Section, with an alternative weighted formulation.

3.1.1 Default Item Voting

The strategy of averaging the ratings of a particular user, and setting all his unrated items to this mean value has proven in the past to be a successful approach [34]. As discussed briefly in the previous section, it does assume that a user rates all items rather similarly.

While this assumption might hold in the cases where a user chooses to only rate items he either likes or dislikes, it does not hold in a mixed scenario. Moreover, in cases where a user did rate only similar liked or not liked items, when such an average value is extended to *all* unrated items, it stands to reason that the assumption will continue to hold; which rarely does.

One idea to pursue is to compute a default voting across the items dimension, instead of the users one. Under this point of view, when we have to provide a default value to an unrated item of a given user, we do not compute the mean of the items rated by that user, but instead compute the mean of the ratings given to that item by other "selected" users.

If we were to compute the average of the rating given to an item by *all* users, undoubtedly we would be achieving little, since there would be high ratings from users that liked the item, mixed with low ratings from users that disliked it. At the end we would be settling for a neutral value in almost all cases but the ones that are either favoured by all, or despised by all.

How to select the neighbourhood of users for which we should compute the item average voting is an open question with no definite answer. In this Thesis we chose to run the unflavoured formulation of the Collaborative Filtering algorithm to produce a neighbourhood of similar candidates.

By unflavoured formulation we refer to the basic Collaborative Filtering algorithm to which no enhancement has been added. Under this algorithm, only items rated by target and candidate count towards the similarity computation. All other rated elements are simply discarded.

The item average algorithm was subsequently carried out as a second step, over the neighbourhood of similar elements. The underlying assumption here is that users or items belonging to the similar neighbourhood would rate the unrated items similarly. While not strictly true, it should hold better than averaging regardless of similarity.

The algorithm resulting from this approach looks as follows:

```
Default Item Voting User Based Collaborative Filtering Algorithm

 1  **Input**: User-Item Rating matrix $R$
 2  **Output**: Recommendation lists of size $l$
 3  **Const** $l$: Number of items to recommended to user $u$
 4  **Const** $v$: Maximum number of users in $\mathcal{N}(u)$, the neighbours of user $u$
 5  **For** each user $u$ **Do**
 6      **For** each common item $i$ that users $u$ or $v$ have not rated **Do**
 7          **Set** rating of $i$ to the average of all ratings of this item in $\mathcal{N}_i(u)$
 8      **End**
 9      **Set** $\mathcal{N}(u)$ to the $v$ users most similar to user $u$
10      **For** each item $i$ **Do**
11          Combine ratings given to item $i$ by user $u$ and neighbours $\mathcal{N}_i(u)$
12      **End**
13      Recommend to user $u$ the top-$l$ items with the highest predicted rating $\hat{r}_{ui}$
14  **End**
```

The Default Item Voting Algorithm is very similar to standard Collaborative Filtering Algorithm described in Section (2.6.1). The main difference between them is found where all unrated items are set to an average of ratings from all candidate users that are deemed similar to the target user (Lines 6-8).

3.1.2 Indirect Estimation

Attempting to set all unrated items to a default value, while fixing the difficult sparsity problem, does it by flattening out the variance in a user's ratings. After filling all unrated preferences with the same value, the distribution becomes quite uniform. A better strategy would attempt to predict each one of the missing ratings individually.

The idea proposed here is to approach the recommendation as a two step process. We commence as usual, matching a target and a candidate in order to compute their similarity. However, instead of filtering out the ratings where one of the two items is missing, we estimate them before proceeding. With the estimates computed, and the sparsity problem fixed, we continue the calculation of the similarity, but now with a full set of ratings.

The underlying paradigm used by the Indirect Estimation is truly borrowed from computer science where many problems are solved by applying one level of indirection; for an item missing a rating, we propose to halt the process, estimate the missing rating using a standard formulation (akin to an indirection, since it kicks off a separate, out of the main workflow estimation), and only then proceed as normal.

In algorithm form, the idea proposed takes the following form:

Indirect Estimation User Based Collaborative Filtering Algorithm

```
1  **Input**: User-Item Rating matrix $R$
2  **Output**: Recommendation lists of size $l$
3  **Const** $l$: Number of items to recommended to user $u$
4  **Const** $v$: Maximum number of users in $\mathcal{N}(u)$, the neighbours of user $u$
5  **For** each user $u$ **Do**
6      **For** each common item $i$ that users $u$ or $v$ have not rated **Do**
7          **Set** rating of $i$ to estimate computed by Section (2.6.1)
8      **End**
9      **Set** $\mathcal{N}(u)$ to the $v$ users most similar to user $u$
10     **For** each item $i$ **Do**
11         Combine ratings given to item $i$ by user $u$ and neighbours $\mathcal{N}_i(u)$
12     **End**
13     Recommend to user $u$ the top-$l$ items with the highest predicted rating $\hat{r}_{ui}$
14 **End**
```

When contrasted to the standard Collaborative Filtering Algorithm from Section (2.6.1) it is seen that the Indirect Estimation Algorithm adds an extra computational step (Lines 6-8). In this portion of the algorithm, any unrated item is first estimated before being used to find similar users. This ensures that the sparse matrix of User-Item Ratings is dense, but the values used to not fill it are not mere averages but the best estimates available.

Observing that all of the candidates' unrated entries need to be estimated for every given prediction, suggests that in fact this strategy could be split into a two step approach, where one of them could in fact be computed offline, as a preprocessing step, to be used during the online estimation.

This first step would involve estimating *all* unrated entries of the Sparse matrix, generating a dense matrix as a starting point for the Recommender Algorithm. One major advantage of this split is that the estimation itself is an expensive process, but that now takes place much before the online recommendation, so it does not impact the critical moment when the algorithm is used. However, since the User-Item Ratings matrix in now densely populated with a substantial increase in samples, the algorithm requires a longer Processing Time to complete.

The full estimation of the User-Item Ratings matrix has though one drawback, and it involves changes in its data by the addition of new ratings. In principle, these new ratings could simply replace any estimated ones and leave the others as-is. However, after several new ratings have been added it would be prudent to recompute all estimated values using the newly added ratings.

3.1.3 Class Type Grouping

The Class Type Grouping proposal is inspired from Context Aware algorithms, where extra features included in higher dimensions are also used as part of the recommendation. This algorithm assumes that every item is of one of c classes, and will effectively segment the User-Item Ratings matrix into c smaller sub-matrices where only equal-class items will be considered during the similarity computation.

The base assumption of Collaborative Filtering algorithms, namely that similar users will like similar items, is hard to use effectively when the set of items belong to many different classes. Two different user might agree on a given item class, for example action movies in the case of a Movie Recommender, but one of them might like drama while the other one dislike it. If we were only interested in recommending action movies, why consider drama movies at all?

If we further add one more genre, like comedy, the picture becomes even harder to accommodate. For an effective similarity we would now need them to agree on action, drama *and* comedy.

The Class Type Grouping strategy attempts to compare action to action, drama to drama, and comedy to comedy, and ensure that only the target class sought is matched for similarity, disregarding the other items which might be agreed on or not.

One notable drawback of this strategy is that it excludes preferences from the User-Item Ratings matrix, effectively reducing the data available for processing. However, since the algorithm is focusing the similarity search to samples belonging to the same class, it is expected to compensate for this, as it filters out the noise introduced by the other controvertible classes.

In pseudo-code, this approach takes the following form:

Class Type Grouping User Based Collaborative Filtering Algorithm

```
1  **Input**: User-Item-Class Rating matrix $R$, and item class to recommend $c$
2  **Output**: Recommendation lists of size $l$
3  **Const** $l$: Number of items to recommended to user $u$
4  **Const** $v$: Maximum number of users in $\mathcal{N}(u)$, the neighbours of user $u$
5  **For** each user $u$ **Do**
6      **For** each item $i$ **Do**
7          **If** item is *not* of class $c$
8              exclude the item $i$ column from the input matrix $R$
9          **End**
10     **End**
11     **Set** $\mathcal{N}(u)$ to the $v$ users most similar to user $u$
12     **For** each item $i$ that user $u$ has not rated **Do**
13         Combine ratings given to item $i$ by neighbours $\mathcal{N}_i(u)$
14     **End**
15     Recommend to user $u$ the top-$l$ items with the highest predicted rating $\hat{r}_{ui}$
16 **End**
```

The Class Type Grouping Algorithm is identical to the standard Collaborative Filtering Algorithm from Section (2.6.1) except for the addition of a filtering step (Lines 6-10). In this part of the algorithm, any item that has a class different than the class c we are trying to recommend, the item is filtered out. The remaining items are subsequently fed to the standard Recommender Algorithm as customary.

3.1.4 Weighted Ensemble

An ensemble of algorithms for Recommender Systems is nothing new. In fact, the Netflix.com competition of 2006 was won by a team employing an ensemble of over 100 different strategies to produce recommendations [31].

The proposal put forward here is to consider a weighted ensemble of a standard Collaborative Filtering algorithm (with or without improvements) and a Class Type Grouping algorithm. The immediate drawback is that every estimation needs to be processed by two separate algorithms, however, they could be computed in parallel since they are completely independent.

In pseudo-code, this approach takes the following form:

```
Weighted Ensemble User Based Collaborative Filtering Algorithm
1 Input: User-Item-Class Rating matrix $R$, and item class to recommend $c$
2 Output: Recommendation lists of size $l$
3 Const $l$: Number of items to recommended to user $u$
4 For each user $u$ Do
5     Set $n_s$ to the neighbourhood computed by Section (2.6.1)
6     Set $n_g$ to the neighbourhood computed by Section (3.1.3)
7     Set $n$ to the combination of ratings given by $n_s$ and $n_g$
8     Recommend to user $u$ the top-$l$ items with the highest rating from $n$
9 End
```

This algorithm, as shown by the pseudo-code, uses a combination of algorithms from Section (2.6.1) and Section (3.1.3) to output a combined estimation (Lines 5-7) which is then sorted, and cut at the l threshold to recommend to the user.

The reasoning behind this approach is that Class Type Grouping can be a useful strategy, but could prove too restrictive. Generating a formulation that makes use of a proportion of such algorithm, with a proportion of a broader one, like a typical Collaborative Filtering approach could help fine tune the latter one without restricting the recommendations too much.

Class Type Grouping is anchored on the observation that people agreeing with one particular class need not necessarily agree on other classes as well. And if we incorporate these other classes to the similarity, we could very well spoil our similarities because of the added noise.

However, if some proportion of users do agree on more than one class, this approach would not find them, and clearly these are *truly* similar users, since they agree on more than one front. Arguably, having an algorithm that would weight both approaches could prove very sensible.

Mathematically, the Weighted Ensemble strategy is formulated as follows:

$$\hat{r}_{WE} = \alpha \cdot \hat{r}_{CF} + (1 - \alpha) \cdot \hat{r}_{CTG} \tag{3.1}$$

where $\hat{r}_{WE}$ is the estimated rating of the Weighted Ensemble, α is a suitable weight, $\hat{r}_{CF}$ is the estimated rating obtained with the Collaborative Filtering, and $\hat{r}_{CTG}$ is the estimated rating obtained by the Class Type Grouping.

3.2 Item Based Formulations

The four strategies put forward in this Chapter, and detailed in Sections (3.1.1, 3.1.2, 3.1.3 and 3.1.4) have been presented for the User Based formulation of the Memory Based Collaborative Filtering Algorithm, however, similar formulations have been derived for the Item Based approach studied herein. The only difference between both algorithms is that the User Based looks at similarities between users, while the Item Based approach looks for similarities between items as rated by users. Pseudocode for the standard Item Based Collaborative Filtering would look as follows:

Standard Top N-Neighbours Item Based Collaborative Filtering Algorithm

```
1  Input: User-Item Rating matrix $R$
2  Output: Recommendation lists of size $l$
3  Const $l$: Number of items to recommended to user $u$
4  Const $j$: Maximum number of items in $\mathcal{N}(i)$, the neighbours of item $i$
5  For each item $i$ Do
6      Set $\mathcal{N}(i)$ to the $j$ items most similar to item $i$
7      For each user $u$ that has no rating for item $i$ Do
8          Combine ratings of user $u$ in neighbours $\mathcal{N}_u(i)$
9      End
10     Recommend to user $u$ the top-$l$ items with the highest predicted rating $\hat{r}_{ui}$
11 End
```

All Item Based formulations of the algorithms proposed in this Thesis would follow the pseudo-code above, with their relevant additions and modifications in place.

3.3 Summary

In this section we have described four different proposals for improving Collaborative Filtering Recommender System algorithms. The proposals put forward concentrated on two things: the strong weakness of the general strategy, namely the sparsity of User-Item Rating matrix brought about the fact that users tend to rate a small proportion of the items, and on the under-utilized item class information, which was argued that is generally available in most datasets, but very seldom used.

The first strategy, termed Default Item Voting modified the Default User Voting approach which sets all unrated items of a given user to the average of all the user's items. Our proposal, deviates from this approach in that the unrated item is set to the average of the given item as rated by all similar user candidates. The assumption underlying this strategy is that similar user candidates would tend to rate a particular item similarly, and thus the average rating could be a good default value to use, to fix the sparsity problem.

The second strategy, termed Indirect Estimation proposed to actually compute every single unrated item encountered. Instead of setting all missing preferences to a default value, the argument put forward was that estimating them should yield better results. It was noted that estimating all unrated preferences is a computationally intensive task, but in our formulation it can be done offline, resulting in an online recommendation of comparable complexity to any other Collaborative Filtering approach.

The third strategy, termed Class Type Grouping sought to make use of extra information generally available with modern datasets, namely the class type of the item. In cases where the system is after recommending a particular type of item, this approach might work well since it would filter out all outlier items and concentrate on finding similar elements to the class in question.

The fourth and last strategy, termed Weighted Ensemble formulated an equation that incorporated the standard Collaborative Filtering algorithm, and the Class Type Grouping. The rational behind this approach was that in some cases the Class Type Grouping might prove to be too restrictive, and assembling its result with a proportion of a standard Collaborative Filtering might be beneficial.

The results of carrying out experiments with each of the four strategies, together with more standard formulations will be reported in the next section of this Thesis.

Chapter 4

Experiments

4.1 Data

In order to test the proposal put forward in this Thesis, we have chosen to use the MovieLens 100k [40] dataset, which is a very popular dataset in the context of Recommender Systems. It has been widely used in research, and baseline figures are readily available for comparative purposes [41].

The dataset comprises a total of 100,000 ratings from a pool of 943 users, voicing preferences on 1682 movies. The rating scale is a typical 1-5 integer number, 1 being a low rating, and 5 being a high one. An advantage of this dataset over others is that it also includes an extra file with tag information about the movies, which allows us to incorporate them into our Class Type Grouping proposal of Section (3.1.3).

GroupLens, the Social Computing research group at the University of Minnesota, offers two other complementary datasets to the MovieLens 100k, namely the MovieLens 1M, and the MovieLens 10M. They have not been used in this thesis due to their size, but would be an interesting exercise for future work, particularly leveraging the Hadoop [5] algorithms for parallel computations.

The dataset comes as a single list of 100k entries of triple values, separated by commas (CSV file format), representing a user Id, an item Id, and the rating. It also comes split into 5 sets of base and test files, with 4/5 and 1/5 partitions on each pair. A separate file provided with the pack, includes the categories of each movie, which can be any one, or a combination from the following list:

0. unknown
1. Action
2. Adventure

3. Animation
4. Children's
5. Comedy
6. Crime
7. Documentary
8. Drama
9. Fantasy
10. Film-Noir
11. Horror
12. Musical
13. Mystery
14. Romance
15. Sci-Fi
16. Thriller
17. War
18. Western

An example of the categories of a movie is:

```
1|Toy Story (1995)|01-Jan-1995|...|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0
```

Which implies that *Toy Story* corresponds to the genres of *Animation*, *Children's* and *Comedy*.

An efficient way to code the multi-category nature of a film is to use bit-coding. Under this schema, *unknown* would be 1, *Action* would be 2, *Adventure* would be 4, *Animation* would be 8, and so forth in powers of 2. For example, *Toy Story* would have a category of 56, since $Animation + Children's + Comedy = 8 + 16 + 32 = 56$. This format is very convenient since finding two movies with similar categories reduces to the simple boolean algebra operation of AND.

4.2 Methodology

The state of the art Apache Mahout Collaborative Filtering Recommender System [6] was chosen as the basis for the implementation of our proposals. Mahout is not only a robust Recommender System, but it is part of the Apache foundation, which makes it available as open source, meaning that the code is freely distributable.

Having Mahout as a basis for our algorithms meant that we could leverage the power of this a Recommender System—like reading a file, selecting suitable neighbourhoods, computing similarities, producing an estimate—and concentrate entirely on the additions we intended to implement and test, namely the four proposals laid out in Chapter (3).

The Mahout implementation comes already with Euclidean Distance, Pearson Correlation, and Cosine Distance similarity functions among others, Top N-Neighbours or Threshold Filtering functions for neighbourhood selection, User Default Voting and Significance Weighting for missing rating inference, and User and Item Based strategies. With the exception of Threshold Filtering, we have used all other options listed, to provide a clear baseline metric for comparison.

The Top N-Neighbours requires a suitable maximum neighbourhood size to be specified. This parameter was obtained by cross-validating the dataset for a range of neighbourhood sizes, on User and Item Based, and on each of the similarity functions separately, under a basic scenario (no missing rating inference or other improvement). Suitable maximum neighbourhoods were also derived for our Class Type Grouping proposal. This resulted in the following maximum neighbourhood sizes:

	User Based		Item Based	
	Plain	Grouped	Plain	Grouped
Euclidean Distance	150	200	250	80
Pearson Correlation	310	250	325	125
Cosine Distance	340	400	300	150

Table 4.1: Neighbourhood size per Similarity Function and Algorithm

Maximum neighbourhood sizes for each basic setup were chosen to minimize the global Accuracy, but not to maximize the global Coverage. An illustration of the effect of neighbourhood choice on Accuracy and Coverage is shown in Figure (4.1), where we see two important areas. In the first, both Accuracy and Coverage improve, until Accuracy reaches its minimum point. From there onwards, Coverage continues to improve, but Accuracy actually worsens. The neighbourhood size at the point of inflection was chosen as the maximum neighbourhood value.

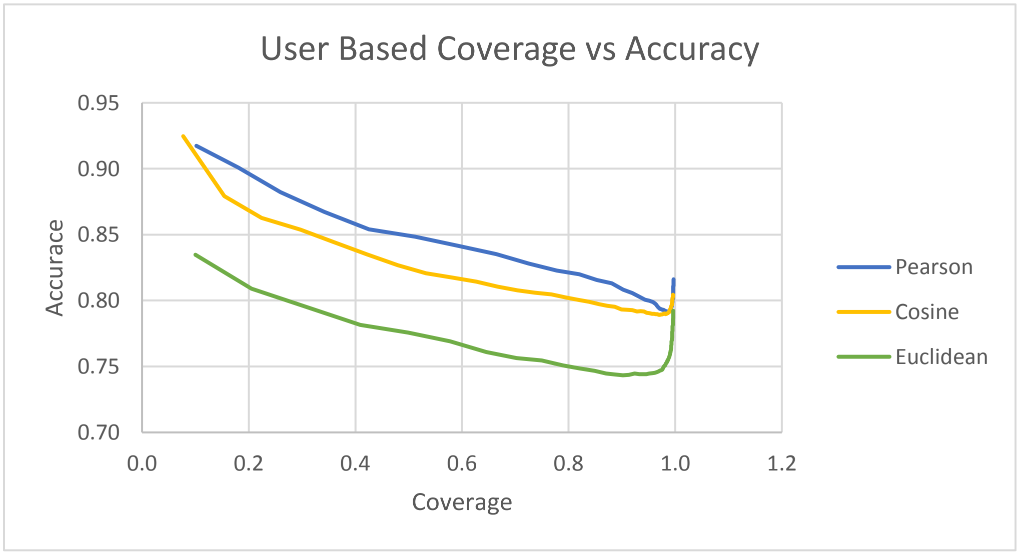


Figure 4.1: User Based Coverage vs Accuracy progression

Accuracy of an entry was calculated using the leave-one-out cross-validation strategy (removing a sample from the dataset, and estimating its rating based on the remaining samples). Global Accuracy was computed using the individual accuracies obtained. Coverage was computing by averaging the number of estimations that were computed. MAE and $RMSE$ values were produced as a measure of Accuracy of each algorithm following the equations of Section (2.7.2).

4.3 Description

The basic formulations included in the Mahout Recommender System, and used herein, are the Euclidean Distance, the Pearson Correlation, and the Cosine Distance similarity functions. All these three similarities can be used with a User Based or an Item Based approach. Neighbourhood selection was confined to the N-Top Neighbours, where the maximum N was established by cross-validation.

Missing ratings can be inferred by means of Default User Voting, and all these algorithms can be weighted by applying a Significance Weighting, which renders candidates sharing more items in common, more important than others sharing fewer items in common. All these Mahout *out-of-the-box* strategies were run as baseline.

In Chapter (3) we put forward four different proposals for improving the Accuracy of Memory Based Collaborative Filtering algorithms. The first proposal involved replacing missing item ratings with an average of the selected neighbourhood's item rating, and we called it Default Item Voting.

The second proposal consisted on estimated missing values while estimating a given preference, effectively introducing an extra *aside* computation step, which we called Indirect Estimation. During the implementation of this algorithm it was discovered that it could be coded as an offline processing step, and it was termed Prepare Indirect. The Indirect Estimation then makes use of the *prepared* values whenever a missing rating is found.

The third proposal involved using the extra information available in the dataset, in the form of item categories, augmenting the typical user-item matrix to a third dimension of labels. We termed this strategy Class Type Grouping.

Lastly, we proposed using a weighted equation of a Default Estimation (non-class discriminative) and a Class Type Grouping formulation, we called this strategy Weighted Ensemble. In this case, there is the proportionality parameter that need to be decided. A preliminary feel for the parameter suggested that a simple weighting with no cut-off would fair well, and this was the strategy chosen. There is ample place to explore different settings in future work.

Among all the proposals, the first two strategies run sideways to the basic formulations. Strategy three, however, is really a different way to run all the previous strategies, and thence doubles the number of algorithms. Lastly, strategy four is a combination of the first two strategies, and the third. In numbers, we have evaluated a total of 144 different strategies, which we will report in the next section of this Thesis.

As stated earlier, Apache Mahout was used as the basis for our own implementations. Mahout is written in Java, and since the source code is openly available, it can be changed at will to add any new algorithms. All four strategies required new functionality that was not present in the application, and was added as necessary.

While Mahout does provide an Item Average Inferrer, it is not a match to our own formulation, and thus it was added to the software. Likewise, the Indirect Estimation algorithm, and the Class Type Grouping are novel, and were added to Mahout. The Weighted Ensemble, being a weighted average of previous results, was computed entirely in Microsoft Excel.

4.4 Results

The results of the various algorithms tested in the thesis are tabulated next. They include those that were run using *out-of-the-box* Mahout functionality, and those that were added, following our proposals. Baseline values are the ones obtained by using the Plain strategy with the different similarity functions, under User and Item Based approaches. This Plain strategy makes no inference of missing preferences.

The first set of tables show four basic strategies: Plain, Plain with Significance Weighting (Plain SW), Default User Voting (User), Default User Voting with Significance Weighting (User SW); and the first two of our proposed strategies, with and without Significance Weighting: Default Item Voting (Item), Default Item Voting with Significance Weighting (Item SW), Indirect Estimation (Ind), and Indirect Estimation with Significance Weighting (Ind SW). Time is in minutes.

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.7433	0.7487	0.7709	0.7867	0.7429	0.7487	**0.7251**	0.7662
RMSE	0.9499	0.9538	0.9730	0.9908	0.9492	0.9533	0.9305	0.9724
Cov %	0.9019	0.9536	0.9422	0.9907	0.9019	0.9536	0.9389	0.9850
Time	2.8	3.6	16.0	15.7	4.1	5.1	58.0	60.0

Table 4.2: User Based with Euclidean Distance similarity, Default Estimation

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.7870	0.7596	0.7373	0.8185	0.7814	0.7574	**0.6760**	0.8058
RMSE	1.0072	0.9723	0.9444	1.0410	1.0004	0.9693	0.8832	1.0338
Cov %	0.8551	0.9416	0.9584	0.9985	0.8551	0.9416	0.8493	0.9952
Time	3.2	4.0	20.0	21.2	4.3	5.4	77.1	81.3

Table 4.3: Item Based with Euclidean Distance similarity, Default Estimation

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.7913	0.7916	**0.7883**	0.8017	0.8005	0.7986	0.7925	0.7982
RMSE	0.9977	0.9972	0.9906	1.0047	1.0100	1.0060	1.0005	1.0045
Cov %	0.9863	0.9889	0.9919	0.9945	0.9863	0.9889	0.9829	0.9907
Time	3.2	4.0	16.0	16.7	7.2	6.7	66.9	67.5

Table 4.4: User Based with Pearson Correlation similarity, Default Estimation

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.8403	0.8133	**0.7592**	0.8193	0.8459	0.8174	0.8274	0.8275
RMSE	1.0512	1.0225	0.9585	1.0370	1.0604	1.0302	1.0393	1.0433
Cov %	0.9732	0.9847	0.9967	0.9979	0.9732	0.9847	0.9596	0.9966
Time	3.4	4.4	20.5	20.9	6.2	5.3	83.8	86.4

Table 4.5: Item Based with Pearson Correlation similarity, Default Estimation

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.7891	0.7880	0.8162	0.8066	0.7893	**0.7881**	0.7994	0.7963
RMSE	1.0043	1.0029	1.0399	1.0255	1.0045	1.0030	1.0183	1.0120
Cov %	0.9712	0.9748	0.9710	0.9850	0.9712	0.9748	0.9692	0.9818
Time	3.8	3.7	15.4	15.9	5.7	5.8	65.7	66.0

Table 4.6: User Based with Cosine Distance similarity, Default Estimation

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.9634	0.9397	0.9045	**0.8588**	0.9633	0.9396	1.0597	1.0309
RMSE	1.2217	1.1953	1.1690	1.1072	1.2216	1.1953	1.3571	1.3161
Cov %	0.6179	0.6519	0.9420	0.9975	0.6179	0.6519	0.5033	0.6527
Time	3.9	4.3	20.6	20.4	4.4	4.6	70.6	71.5

Table 4.7: Item Based with Cosine Distance similarity, Default Estimation

	User Based			Item Based		
	Euclidean	Pearson	Cosine	Euclidean	Pearson	Cosine
Time	43.0	41.2	44.1	36.4	36.4	37.2

Table 4.8: Prepare Indirect offline algorithm, Default Estimation

Tables (4.2) to (4.7) illustrate the results obtained by the different algorithms under a Default Estimation, which refers to the estimation carried out with all the samples, regardless of which class they belong to. This is the typical estimation employed in most Recommender Systems. Bold measures represent the best MAE error readings obtained in each case. Table (4.8) shows the Time required to process the offline step of the Indirect Estimation algorithm.

We tabulate next the results obtained by using our proposed third algorithm, the Class Type Grouping.

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.7511	0.7510	0.7551	0.7559	0.7501	0.7502	**0.7278**	0.7347
RMS	0.9570	0.9559	0.9567	0.9581	0.9557	0.9546	0.9315	0.9390
Cov %	0.9325	0.9510	0.9497	0.9712	0.9325	0.9510	0.9295	0.9572
Time	19.0	18.8	120.2	119.8	23.7	22.4	39.8	34.6

Table 4.9: User Based with Euclidean Distance similarity, Class Type Grouping

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.7756	0.7615	0.7351	0.8199	0.7627	0.7738	**0.6592**	0.7512
RMS	0.9990	0.9799	0.9450	1.0487	0.9811	0.9968	0.8677	0.9749
Cov %	0.6874	0.8000	0.8837	0.9883	0.8000	0.6874	0.6919	0.8663
Time	27.1	28.9	204.1	214.4	28.0	29.3	71.4	62.2

Table 4.10: Item Based with Euclidean Distance similarity, Class Type Grouping

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.8075	0.8068	0.7989	0.8003	0.8112	0.8097	**0.7909**	0.7916
RMS	1.0216	1.0206	1.0067	1.0076	1.0258	1.0236	1.0012	1.0014
Cov %	0.9732	0.9752	0.9810	0.9834	0.9732	0.9752	0.9708	0.9743
Time	19.6	19.4	122.5	124.0	29.9	26.2	41.2	35.6

Table 4.11: User Based with Pearson Correlation similarity, Class Type Grouping

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.8306	0.8087	**0.7559**	0.8137	0.8146	0.8342	0.7649	0.7829
RMS	1.0504	1.0266	0.9626	1.0372	1.0327	1.0534	0.9647	0.9932
Cov %	0.9042	0.9297	0.9814	0.9910	0.9297	0.9042	0.9061	0.9552
Time	28.1	28.4	206.0	212.4	30.2	30.0	64.8	62.9

Table 4.12: Item Based with Pearson Correlation similarity, Class Type Grouping

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.7943	**0.7941**	0.8069	0.8057	0.7944	0.7942	0.7987	0.7983
RMS	1.0083	1.0078	1.0272	1.0251	1.0085	1.0079	1.0156	1.0143
Cov %	0.9715	0.9728	0.9744	0.9782	0.9715	0.9728	0.9707	0.9745
Time	19.4	19.6	121.9	126.9	27.0	26.4	39.9	35.6

Table 4.13: User Based with Cosine Distance similarity, Class Type Grouping

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.8462	**0.8385**	0.8775	0.8401	0.8395	0.8470	0.8775	0.8731
RMS	1.0812	1.0731	1.1318	1.0804	1.0743	1.0823	1.1298	1.1216
Cov %	0.6920	0.7120	0.9101	0.9855	0.7120	0.6920	0.7228	0.7790
Time	28.4	28.7	206.9	214.6	29.3	29.4	63.0	64.3

Table 4.14: Item Based with Cosine Distance similarity, Class Type Grouping

	User Based			Item Based		
	Euclidean	Pearson	Cosine	Euclidean	Pearson	Cosine
Time	59.4	63.6	54.0	38.4	42.2	40.9

Table 4.15: Prepare Indirect offline algorithm, Class Type Grouping

Tables (4.9) to (4.14) illustrate the results obtained by the Class Type Grouping algorithm under the different scenarios tested, with Table (4.15) reflecting the Time required to process the offline step of the Indirect Estimation algorithm.

Lastly, we tabulate the results obtained by our fourth algorithm, the Weighted Ensemble. No Processing Time is given because the values were computed with Excel, and are readily available.

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.7342	0.7325	0.7440	0.7483	0.7358	0.7322	**0.7169**	0.7421
RMS	0.9338	0.9307	0.9396	0.9442	0.9364	0.9301	0.9170	0.9439
Cov %	0.8858	0.9014	0.8813	0.9014	0.9019	0.9014	0.9102	0.9545

Table 4.16: User Based with Euclidean Distance similarity, Weighted Ensemble

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.7612	0.7439	0.7239	0.8134	0.7579	0.7432	**0.6395**	0.7638
RMS	0.9690	0.9487	0.9248	1.0365	0.9647	0.9478	0.8292	0.9808
Cov %	0.6535	0.7864	0.8786	0.9883	0.6535	0.7864	0.6543	0.8659

Table 4.17: Item Based with Euclidean Distance similarity, Weighted Ensemble

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.7928	0.7905	0.7867	0.7982	0.8000	0.7975	**0.7848**	0.7881
RMS	0.9983	0.9954	0.9884	1.0002	1.0092	1.0044	0.9905	0.9929
Cov %	0.9703	0.9862	0.9855	0.9860	0.9863	0.9862	0.9668	0.9730

Table 4.18: User Based with Pearson Correlation similarity, Weighted Ensemble

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.8212	0.8023	**0.7514**	0.8109	0.8336	0.8088	0.7792	0.7942
RMS	1.0272	1.0095	0.9502	1.0282	1.0464	1.0199	0.9749	1.0013
Cov %	0.8990	0.9731	0.9725	0.9732	0.9732	0.9731	0.8944	0.9548

Table 4.19: Item Based with Pearson Correlation similarity, Weighted Ensemble

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	**0.7855**	0.7859	0.8037	0.7969	0.7875	0.7860	0.7943	0.7931
RMS	0.9969	0.9979	1.0226	1.0125	1.0001	0.9980	1.0099	1.0070
Cov %	0.9610	0.9711	0.9575	0.9670	0.9712	0.9711	0.9594	0.9696

Table 4.20: User Based with Cosine Distance similarity, Weighted Ensemble

	Plain	Plain SW	User	User SW	Item	Item SW	Ind	Ind SW
MAE	0.8860	0.8762	0.8868	**0.8560**	0.8872	0.8769	0.9744	0.9448
RMS	1.1153	1.1164	1.1408	1.0975	1.1291	1.1172	0.2408	1.2033
Cov %	0.5090	0.6174	0.6022	0.6176	0.6179	0.6174	0.4510	0.5904

Table 4.21: Item Based with Cosine Distance similarity, Weighted Ensemble

Lastly, tables (4.16) to (4.21) illustrate the results obtained by the Weighted Ensemble algorithm. The α parameter in Equation (3.1) was set to 0.5 in all cases.

4.5 Analysis

A plot of the results follow, where we show User Based results as triangles, Item Based results as circles, and the Baseline result as a square, for all similarity functions studied. Good results would tend to be located in the lower right-hand side corner of the graph.

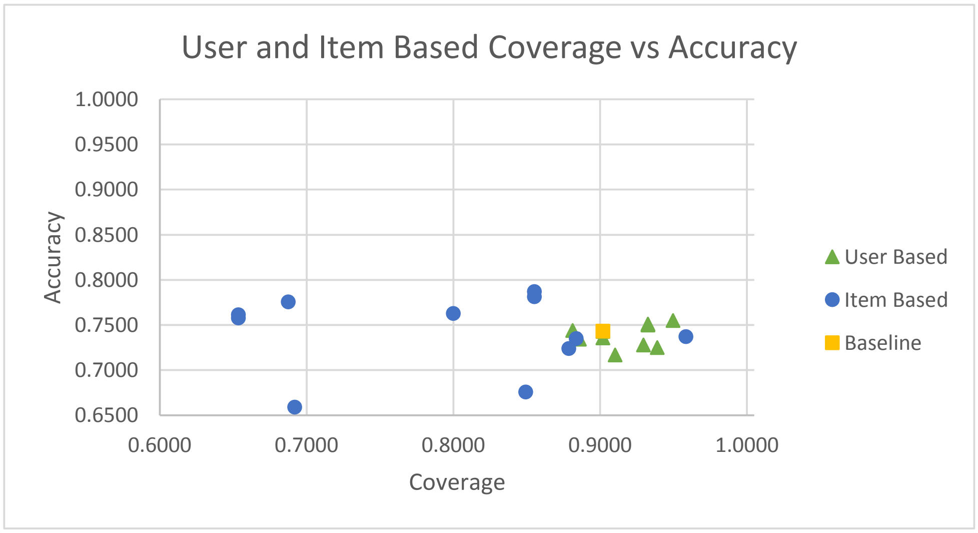


Figure 4.2: User and Item with Euclidean Distance similarity Coverage vs Accuracy

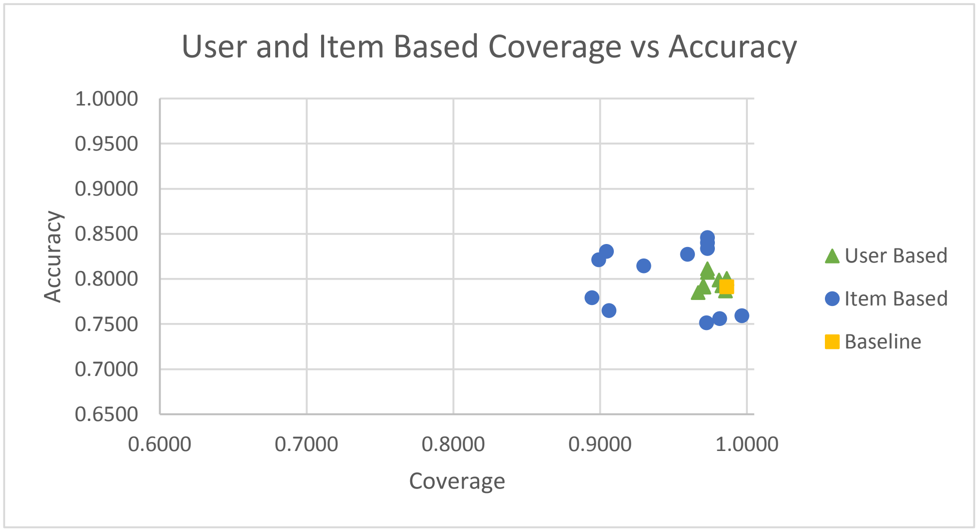


Figure 4.3: User and Item with Pearson Correlation similarity Coverage vs Accuracy

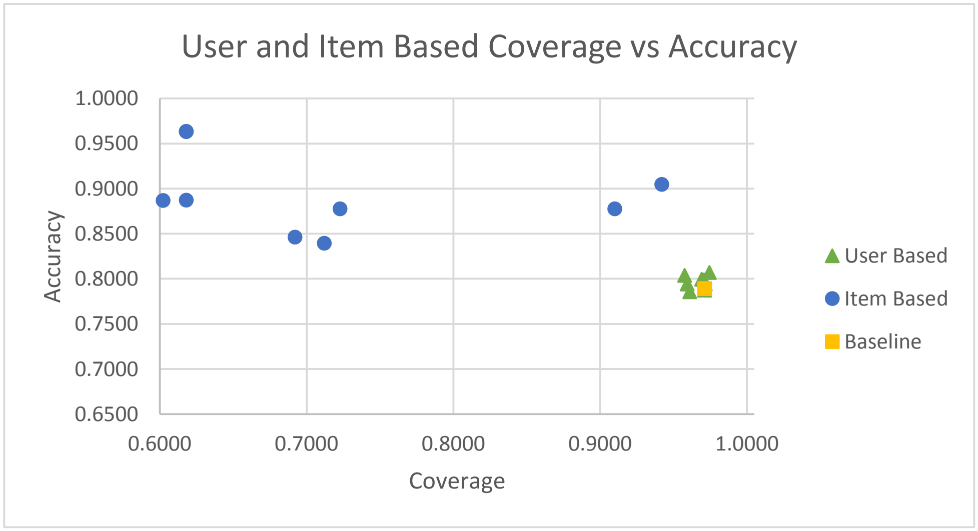


Figure 4.4: User and Item with Cosine Distance similarity Coverage vs Accuracy

In order to carry out a Friedman Test [17] of the results, we re-tabulated the data, capturing only the Accuracies, while disregarding the Coverage obtained by each algorithm. We also defined the *null hypothesis* to be “there is a difference between the algorithms”, and the *alternative hypothesis* to be “there is no difference between the algorithms”, with an *alpha* cutoff value of 0.05.

For the User Based approach, our Accuracy values and Friedman Test results are:

	Euclidean	Pearson	Cosine
1. Plain	0.7433	0.7487	0.7913
2. User	0.7709	0.7867	0.7883
3. Item	0.7429	0.7487	0.8005
4. Indirect	0.7251	0.7662	0.7925
5. Plain Grouped	0.7511	0.7510	0.8075
6. User Grouped	0.7551	0.7559	0.7989
7. Item Grouped	0.7501	0.7502	0.8112
8. Indirect Grouped	0.7278	0.7347	0.7909
9. Plain Weighted	0.7342	0.7325	0.7928
10. User Weighted	0.7440	0.7483	0.7867
11. Item Weighted	0.7358	**0.7322**	0.8000
12. Indirect Weighted	**0.7169**	0.7421	**0.7848**
13. Plain SW	0.7916	0.7891	0.7880
14. User SW	0.8017	0.8162	0.8066
15. Item SW	0.7986	0.7893	0.7881
16. Indirect SW	0.7982	0.7994	0.7963
17. Plain Grouped SW	0.8068	0.7943	0.7941
18. User Grouped SW	0.8003	0.8069	0.8057
19. Item Grouped SW	0.8097	0.7944	0.7942
20. Indirect Grouped SW	0.7916	0.7987	0.7983
21. Plain Weighted SW	0.7905	0.7855	0.7859
22. User Weighted SW	0.7982	0.8037	0.7969
23. Item Weighted SW	0.7975	0.7875	0.7860
24. Indirect Weighted SW	0.7881	0.7943	0.7931

Table 4.22: User Based MAE Accuracy results for all Algorithms

Source	SS	df	MS	Chi-sq	Prob>Chi-sq
Columns	1895.5	23	82.413	37.93	**0.0259**
Error	1552.5	46	33.75		
Total	3448	71			

Table 4.23: User Based results of Friedman Test

For the User Based case, we **accept** the *null hypothesis* since the result is lower than 0.05, meaning that we do find a difference in the algorithms. However, a plot of the intervals reveals that no algorithm has a mean rank significantly different than our baseline, as shown in the User Based Figure (4.5) of the Friedman Test.

For the Item Based approach, our Accuracy values and Friedman Test results are:

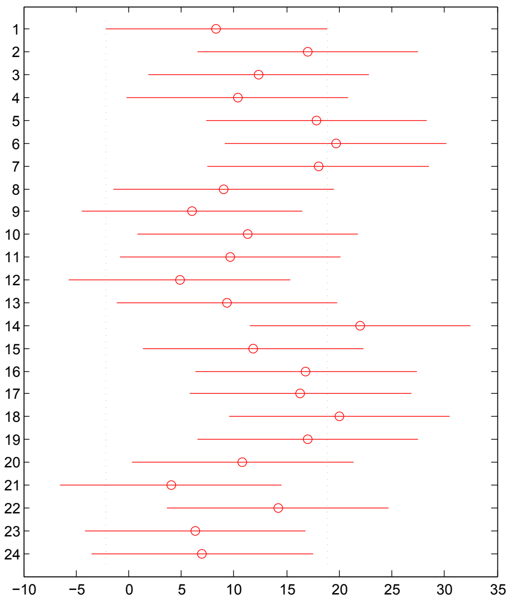
	Euclidean	Pearson	Cosine
1. Plain	0.7870	0.7596	0.8403
2. User	0.7373	0.8185	0.7592
3. Item	0.7814	0.7574	0.8459
4. Indirect	0.6760	**0.6760**	0.8274
5. Plain Grouped	0.7756	0.7615	0.8306
6. User Grouped	0.7351	0.8199	0.7559
7. Item Grouped	0.7627	0.7738	0.8146
8. Indirect Grouped	0.6592	0.7512	0.7649
9. Plain Weighted	0.7612	0.7439	0.8212
10. User Weighted	0.7239	0.8134	**0.7514**
11. Item Weighted	0.7579	0.7432	0.8336
12. Indirect Weighted	**0.6395**	0.7638	0.7792
13. Plain SW	0.8133	0.9634	0.9397
14. User SW	0.8193	0.9045	0.8588
15. Item SW	0.8174	0.9633	0.9396
16. Indirect SW	0.8275	1.0597	1.0309
17. Plain Grouped SW	0.8087	0.8462	0.8385
18. User Grouped SW	0.8137	0.8775	0.8401
19. Item Grouped SW	0.8342	0.8395	0.8470
20. Indirect Grouped SW	0.7829	0.8775	0.8731
21. Plain Weighted SW	0.8023	0.8860	0.8762
22. User Weighted SW	0.8109	0.8868	0.8560
23. Item Weighted SW	0.8088	0.8872	0.8769
24. Indirect Weighted SW	0.7942	0.9744	0.9448

Table 4.24: Item Based MAE Accuracy results for all Algorithms

Source	SS	df	MS	Chi-sq	Prob>Chi-sq
Columns	1280.33	23	55.6667	25.61	**0.3194**
Error	2168.67	46	47.1449		
Total	3449	71			

Table 4.25: Item Based results of Friedman Test

For the Item Based case, we **reject** the *null hypothesis* since the result is higher than 0.05, meaning that we do not find a difference in the algorithms. Also, no algorithm has a mean rank significantly different from our baseline, as shown in the Item Based Figure (4.5) of the Friedman Test.


1..24: Algorithms from Table (4.22)

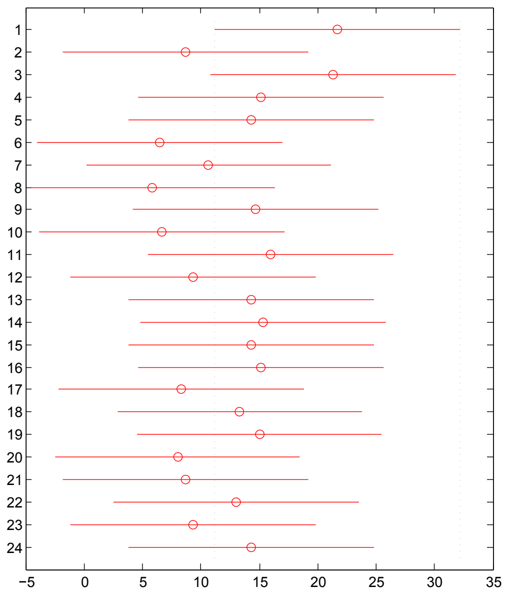

1..24: Algorithms from Table (4.24)

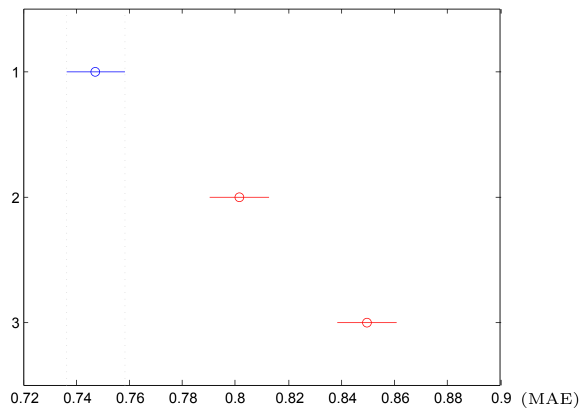
Figure 4.5: Friedman Test of User and Item Based Approaches

Next, we compare each one of our proposals to all the others, by means of ANOVA tests.

We start by looking at the three similarity functions used. In this case, we obtained the following results:

Source	SS	df	MS	F	Prob>F
Groups	0.2511	2	0.1255	58.5402	**0.0000**
Error	0.3024	141	0.0021		
Total	0.5535	143			

Table 4.26: ANOVA Test of Similarity Functions



1: Euclidean Distance, 2: Pearson Correlation, 3: Cosine Distance

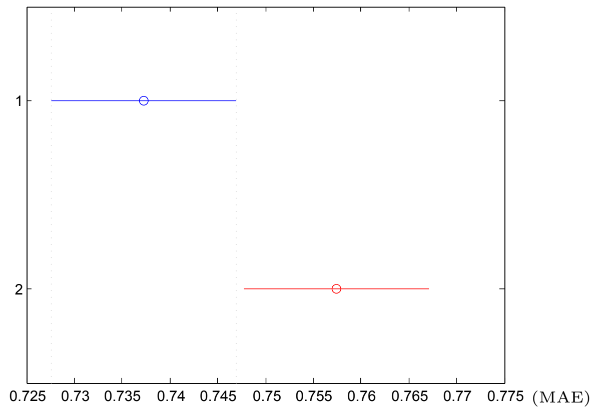
Figure 4.6: ANOVA Test of Similarity Functions

Henceforth we concentrate only on the Euclidean Distance as it has proven to be significantly better on the particular dataset we tested.

We then look at the Significance Weighting strategy, where we obtained:

Source	SS	df	MS	F	Prob>F
Groups	0.0049	1	0.0049	4.4158	**0.0411**
Error	0.0511	46	0.0011		
Total	0.0560	47			

Table 4.27: ANOVA Test of Significance Weighting Algorithm



1: Equal Weighting, 2: Significance Weighting

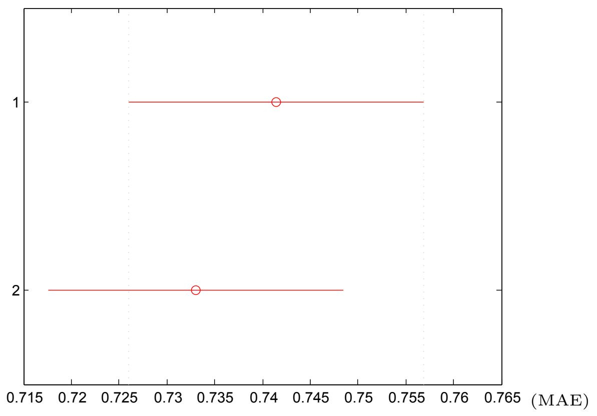
Figure 4.7: ANOVA Test of Significance Weighting Algorithm

Clearly, Significance Weighting performed worse that the default Equal Weighting strategy, and we readily discard it.

We look at User vs Item Based strategies:

Source	SS	df	MS	F	Prob>F
Groups	0.0004	1	0.0004	0.3155	**0.5800**
Error	0.0293	22	0.0013		
Total	0.0297	23			

Table 4.28: ANOVA Test of User vs Item Based Approaches



1: User Based Approach, 2: Item Based Approach

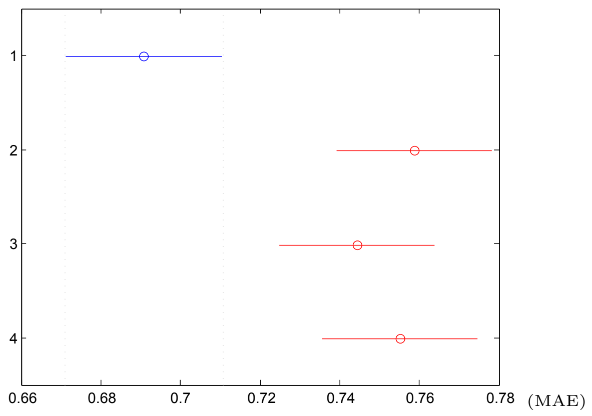
Figure 4.8: ANOVA Test of User vs Item Based Approaches

In the case of User vs Item Based, the results are not significantly different.

We look at the broad algorithms tested, namely, Plain, Default User Voting, Default Item Voting, and Indirect Estimation:

Source	SS	df	MS	F	Prob>F
Groups	0.0180	3	0.0060	10.2013	**0.0003**
Error	0.0117	20	0.0006		
Total	0.0297	23			

Table 4.29: ANOVA Test of Plain, User, Item and Indirect Algorithms



1: Indirect Estimation, 2: Plain, 3: Default User Voting, 4: Default Item Voting

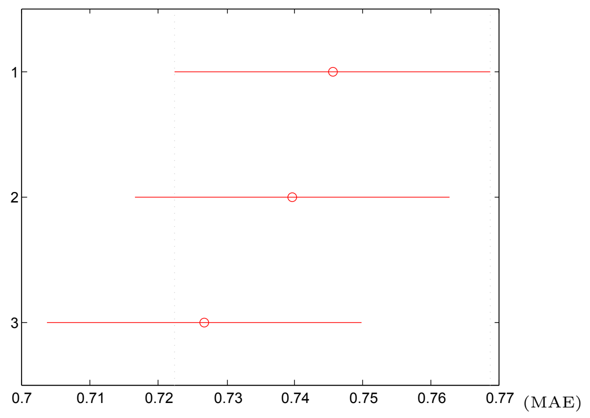
Figure 4.9: ANOVA Test of Plain, User, Item and Indirect Algorithms

The Indirect algorithm performed significantly better than the others.

Lastly, we look at the Grouping and Weighting strategies proposed, against the default strategy:

Source	SS	df	MS	F	Prob>F
Groups	0.0015	2	0.0007	0.5510	**0.5845**
Error	0.0282	21	0.0013		
Total	0.0297	23			

Table 4.30: ANOVA Test of Grouping and Weighting Algorithms



1: Default, 2: Class Type Grouping, 3: Weighted Ensemble

Figure 4.10: ANOVA Test of Grouping and Weighting Algorithms

We do not see significant difference among the three strategies tested.

Lastly, we include a table illustrating how a selection of the algorithms proposed in this Thesis fared when compared to others published [41]:

Source	Algorithm	MAE	Cov %
Thesis	Item Grouped User Based Euclidean	0.7501	0.9325
MyMediaLite	UserItemBaseline	0.7450	-
MyMediaLite	BipolarSlopeOne	0.7446	-
Thesis	Plain User Based Euclidean	0.7433	0.9019
Thesis	Item User Based Euclidean	0.7429	0.9019
MyMediaLite	SlopeOne	0.7404	-
MyMediaLite	UserKNNCosine	0.7370	-
Thesis	Item Weighted User Based Euclidean	0.7358	0.9019
MyMediaLite	UserKNNPearson	0.7281	-
Thesis	Indirect Grouped User Based Euclidean	0.7278	0.9295
MyMediaLite	ItemKNNCosine	0.7270	-
MyMediaLite	FactorWiseMatrixFactorization	0.7252	-
Thesis	Indirect User Based Euclidean	0.7251	0.9389
MyMediaLite	BiasedMatrixFactorization	0.7229	-
MyMediaLite	BiasedMatrixFactorization	0.7221	-
MyMediaLite	BiasedMatrixFactorization	0.7205	-
MyMediaLite	BiasedMatrixFactorization	0.7201	-
MyMediaLite	BiasedMatrixFactorization	0.7194	-
MyMediaLite	SVDPlusPlus	0.7184	-
MyMediaLite	BiasedMatrixFactorization	0.7172	-
MyMediaLite	SigmoidItemAsymmetricFactorModel	0.7170	-
Thesis	Indirect Weighted User Based Euclidean	0.7169	0.9102
MyMediaLite	SVDPlusPlus	0.7155	-
MyMediaLite	SigmoidItemAsymmetricFactorModel	0.7152	-
MyMediaLite	SVDPlusPlus	0.7152	-
MyMediaLite	ItemKNNPearson	0.7144	-
MyMediaLite	SVDPlusPlus	0.7141	-
MyMediaLite	SVDPlusPlus	0.7135	-
MyMediaLite	SVDPlusPlus	0.7130	-
MyMediaLite	SigmoidUserAsymmetricFactorModel	0.7000	-
Thesis	Indirect Item Based Euclidean	0.6760	0.8493
Thesis	Indirect Grouped Item Based Euclidean	0.6592	0.6919
Thesis	Indirect Weighted Item Based Euclidean	0.6395	0.6543

Table 4.31: Comparative of Thesis Accuracy results against others published

Table (4.31) shows the results of some of the algorithms developed in this Thesis, together with others published in the literature. Results for MyMediaLite were carried out using a 5-fold cross validation, as opposed to the leave-one-out cross validation strategy used in our case. They also did not make available the Coverage obtained, reducing merit in the comparison. They are included here as reference only, as it is clear that experiments are not entirely compatible to be fairly compared.

4.6 Discussion

This Thesis proposed four different strategies to improve the performance of Memory Based Collaborative Filtering Recommender Systems. The algorithms proposed and studied here were the Default Item Voting, the Indirect Estimation, the Class Type Grouping, and the Weighted Ensemble. Our premise in this work was that Memory Based approaches still had ample place for improvement, and could compete with the Model Based algorithms, popularized by the 2006 Netflix.com competition.

Tables (4.2) to (4.7) show the results of running the MovieLens 100k dataset on a default Mahout algorithm (Plain), on the Default User Voting inferrer, and on our proposed strategies, the Default Item Voting, and the Indirect Estimation. The tables also show the results of applying a Significance Weighting to each of these four algorithms, which gives a higher importance to samples sharing more points with the target we are trying to estimate. In these set of tables, a User and an Item Based approach has been used, under all three similarity functions, the Euclidean Distance, the Pearson Correlation, and the Cosine Distance.

The baseline value shown in the tables, is the result of the column labeled “Plain”, where we readily see that the Euclidean Distance User Based approach gave a MAE reading of 0.7433 with a Coverage of 0.9019%. Good results would be those that produce a smaller MAE, while keeping the same Coverage, or improving it. Processing Time would ideally not increase by much, but naturally, more complex algorithms will invariable require more computer power.

In table (4.2) we see that the Default Item Voting proposal yielded a marginal improvement over the Plain algorithm, while keeping the same Coverage, and increasing Processing Time by close to 50%. However, the Default Item Voting algorithm (Item) was substantially better than the Default User Voting (User).

The best result of the table is the one obtained by our Indirect Estimation strategy, labeled “Ind”, which gave a MAE of 0.7251 with a Coverage of 0.9389%; that is an Accuracy improvement of 2.5% over the unaltered Mahout Default algorithm, with a further 4% improvement in Coverage. The noticeable drawback was Processing Time which was several orders of magnitude higher.

Significance Weighting (SW) can be seen not to improve the Accuracy results of any algorithm, but improve notoriously the Coverage. In this case, the Default Item Voting performed equally well as the Plain algorithm, with an increase in Processing Time. The Indirect Estimation, combined to the Significance Weighting did not yield a good result, however, it produced a very good Coverage, comparable only to the one obtained by the Default User Voting, albeit with a better MAE.

The Euclidean Distance similarity results obtained by the Item Based approach, tabulated in table (4.3), are also significant. Here we see that the Plain algorithm gave a higher MAE than the User Based counterpart of table (4.2) with a worse Coverage. However, resulting in a similar Coverage, the Item Based Indirect Estimation produced a significantly better MAE of 0.6760, which is in fact the best result in this cross section of results. Comparing this MAE to the one obtained by the User Based Indirect Estimation yields an improvement of 7%. On the other hand, the Coverage was nearly 10% lower, and Processing Time was considerably higher.

In general, we would expect that lower Coverages will give better Accuracies, since difficult samples, likely carrying higher estimated errors, would not be considered. This is why good results, arguably, would have a low Accuracy, with a high Coverage, like the one encountered in table (4.2) by the Indirect Estimation, which was noticeably better than the Plain algorithm tested.

Pearson Correlation and Cosine Distance are two other alternatives to the Euclidean Distance, regularly quoted in the literature. For our dataset, these two similarity functions resulted in worse MAE Accuracies, but significantly better Coverages when compared to the Euclidean Distance. Under these two similarity functions, nor our Default Item Voting, nor our Indirect Estimation proposals yielded better results than the Plain or the Default User Voting strategies.

The next set of tables, (4.9) to (4.14), tabulate the results obtained by our third proposal, the Class Type Grouping. This algorithm made use of class information available in the dataset, to filter out samples when computing the estimations. Our initial expectation of this algorithm was that it would improve the Accuracy of the Recommender System.

The rational behind it was that two users could share a taste for a particular class (for example, *action* genre, in movies), but not on other classes (for example, *comedy*). If we are trying to estimate an action film, looking only at action films while filtering all other genres would ensure that we are only looking for similar tastes across this particular genre, unaffected by discrepancies in other genres.

However, the results, overall, gave a worse MAE than their counterparts without grouping (Default Estimation). It is important to note thought, that in most cases the Coverage improved. And as argued before, a higher Coverage is expected to have a worse Accuracy.

A result worth noting is the one obtained by the Item Based Indirect Estimation of table (4.10). The MAE in this case was 0.6592, which is about 13% better than our baseline of 0.7433. However, this was obtained by the rather low Coverage of 0.6919%.

The last set of tables, (4.16) to (4.21), show the results obtained by our fourth and last strategy, the Weighted Ensemble. This algorithm averaged the results of our first set of tables, and the ones obtained by our third proposal. The idea behind it was that a good compromise perhaps could be achieved by using some weighted measure of a result obtained by estimating with all samples, and estimating with samples of the same class.

Table (4.16) in fact shows that using this strategy produced a improvement in almost all MAE Accuracies. For example, the Plain strategy went down from 0.7433 to 0.7342, and the Indirect Estimation went down from to 0.7251 to 0.7169. Noteworthy is the fact that the Indirect Estimation resulted in a higher Coverage than our baseline, with an Accuracy improvement of about 3.6%.

Table (4.17), which tabulates the results of the Euclidean Distance similarity under an Item Based approach shows also a significant result, namely the one obtained by our Indirect Estimation proposal, with a MAE of 0.6395. That is an improvement in Accuracy of about 16% compared to our baseline. However, the results came at the expense of a significant reduction in Coverage, which only yielded a 0.6543%.

In general, we see that the Euclidean Distance similarity was better than the Pearson Correlation and the Cosine Distance, and that the User Based Indirect Estimation proposal improved the Accuracy results of the Plain strategy without a reduction in Coverage, but a significant increase in Processing Time.

A plot of the results of the various algorithms studied are seen in Figures (4.2) to (4.4). Concentrating our attention on the Euclidean Distance which gave the best overall results, we see that most User Based values are in the vicinity of our baseline. The further they are to the lower-right quadrant of the graph, the better they are, since that means that the Accuracy is lower and the Coverage is higher.

Item Based values tend to be less clustered, and while a few examples display a rather high Accuracy, they do so with a low Coverage. One exception is the Item Based Default User Voting algorithm which improved the Accuracy of our baseline while also improving the Coverage. We see this sample in Figure (4.2) as the rightmost "blue" circle.

Figure (4.3) shows the results of the algorithms when using the Pearson Correlation similarity function. Interestingly, all User and Item Based results with this similarity function tend to be clustered about the baseline, with User Based estimations much closer to the baseline, and Item Based estimation further away.

In the case of the Cosine Distance depicted in (4.4), only User Based results are in the vicinity of the baseline, with all Item Based results being of too poor a quality to consider further.

In order to assess whether the proposed algorithms were significantly better than the basic ones, we subjected the results to a Friedman Test, and to various ANOVA tests. The results of the Friedman Test can be seen in Tables (4.23) and (4.25), for User and for Item Based approaches, respectively. For the case of User Based, our *null hypothesis* stating that "there is a difference between the algorithms" can be seen to hold, however, this was not the case for the Item Based approach.

Figure (4.5) shows the ranges of each algorithm. While the *null hypothesis* in the User Based case was confirmed by calculation, the graph does not appear to show any algorithm being statistically different from any other.

Since the Friedman Test was not conclusive enough, we performed several ANOVA tests to compare different parameters of the strategies studied. First, we looked at the similarity functions used, namely, the Euclidean Distance, the Pearson Correlation, and the Cosine Distance. In this case, Figure (4.6) clearly shows that the three strategies are statistically different, being the Euclidean Distance the best one of the three, and the Cosine Distance, the worst.

An ANOVA test comparing the Significance Weighting algorithm to the default equal sample weighting, shows them in Figure (4.7) to be statistically different, with the default approach being superior in terms of MAE, to the Significance Weighting formulation.

ANOVA tests on User vs Item Based approaches did not show a statistical difference among the two, as depicted in Figure (4.8). This, despite the Item Based approach with Euclidean Distance similarity giving the lowest MAE of all algorithms tested.

Among the four broad algorithm classes used, namely the Plain, the Average User Voting, the Average Item Voting, and the Indirect Estimation, the Indirect Estimation was found to be significantly different, as shown in Figure (4.9).

Lastly, we tested the Default Estimation against the Class Type Grouping and the Weighted Ensemble proposals. In this case, Figure (4.10) shows that there does not seem to be statistical difference among the three algorithms. This, despite the fact that the Weighted Ensemble under Euclidean Distance similarity yielded the best overall result.

Table (4.31) attempts to put the algorithms proposed in this Thesis in context. The table shows published results of other algorithms, performing on the MovieLens dataset used herein. MAE Accuracies obtained by approaches proposed in this Thesis are marked appropriately in the Source column of the table.

While our baseline “Plain User Based Euclidean” occupies position four from the start of the table, our best overall strategy “Indirect Weighted User Based Euclidean” can be found two-thirds down the table. This strategy made use of three of our proposals, packed together, namely, the Indirect Estimation, the Class Type Grouping, and the Weighted Ensemble. The approach lowered Accuracy from the baseline of 0.7433 to 0.7169, a 3.6% improvement, placing it in direct competition with Model Based approaches based on Matrix Factorization techniques. We also point out that if we were to disregard Coverage, then the “Indirect Weighted Item Based Euclidean” triumphed all strategies tabulated.

4.7 Summary

In this section of the Thesis we have presented the methodology used to test the MovieLens dataset chosen, and tabulated the results obtained by the various algorithms studied. In total, we employed the dataset to test 144 different strategies, tabulating the Coverage, Accuracy and Processing Time obtained by each one of them (where applicable).

The results of the algorithms were then analyzed using a Friedman Test, which looks for statistically significant differences in the data. Finer granularity was also sought by means of ANOVA tests of the main parameters of the algorithms tested. Results of the statistical examinations were tabulated with plotted results showing the differences among the strategies looked at.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

This Thesis proposed four strategies aimed at improving the Accuracy of Memory Based Collaborative Filtering Recommender Systems. The four approaches were tested extensively, together with various basic approaches regularly employed in commercial implementations. A total of 144 algorithms were studied, composed of various combinations of our four proposals, and other known algorithms.

The well researched MovieLens 100k dataset was used in all tests. The dataset comprises 100,000 ratings from 943 users, about 1682 movies, and was first compiled by the GroupLens Group of Social Computing Research at the University of Minnesota [40].

In order to test existing approaches, and implement our proposals, the Apache Mahout Collaborative Filtering Recommender System was used [6]. This is an Apache maintained open source project that has become the standard in Recommender Systems for research.

The four different proposals set forward in this Thesis were tested under three different similarity functions, namely the Euclidean Distance, the Pearson Correlation, and the Cosine Distance. Neighbourhood selection was fixed to N-Top Nearest Neighbours, deriving the number of neighbours by cross validation. Estimations of preferences were carried out by leave-one-out cross validation, which removes the testing sample from the set, and uses the remaining samples to estimate it.

User and Item Based approaches were both studied. Other algorithms employed included the Default User Voting and the Significance Weighting; this last one has been applied to all algorithms, to test the effect of the proportionality weighting on the proposals put forward.

Our first algorithm, the Default Item Voting, performed better than its counterpart, the Default User Voting from which it was derived. However, this proposal was fount to be only marginally better than the baseline.

Our second proposal, the Indirect Estimation was shown to be statistically different from other approaches by ANOVA testing, rendering the best results in this Thesis. Used alone, this algorithm improved the baseline by 2.5%. Based on the MAE Accuracy readings obtained, we found that this algorithm learned significantly better the user's preferences than the other neighbourhood selections algorithms tested.

Our third strategy, the Class Type Grouping was expected to perform better than it did. This approach was based on clustering the samples, which is a standard practice in Machine Learning. However, in our case, the clustering was done with complete knowledge of the classes. When applied to the Indirect Estimation it produced results better than our baseline, but worse than Indirect Estimation alone.

Lastly, our fourth strategy, the Weighted Ensemble improved almost all previous Accuracies obtained by the Default Estimation. When applied to the Plain algorithm, it improved the baseline by 1%, and when applied to the Indirect Estimation, it improved the baseline by 3.6% without a loss in Coverage. In the context of this Thesis, the User Based Indirect Estimation Weighted Ensemble algorithm was the best formulation to learn a target user profile, and recommend relevant items.

Overall, our premise that Memory Based Collaborative Filtering Recommender Systems can still be improved, and made to compete with Model Based approaches deriving from Matrix Factorization seems to be supported by this work. We believe that this class of Machine Learning algorithms, which have been in commercial use for the past twenty years, still merit further research.

5.2 Future Work

Several aspects of the algorithms tested here were identified for future research. For example, the Default Item Voting strategy averages the item preferences of the neighbourhood. The effect of neighbourhood size was not tested, having used the value obtained by cross validation from the Plain algorithm.

The implementation of the Indirect Estimation proposal was found to be separable, meaning that it could be carried out in an offline step. This was in fact the strategy followed in this Thesis. What was not studied is the effect of changing preferences on the offline data computed. In a live Recommender System, users are constantly adding and changing ratings. When too many of these values are changed, the offline data would need to be recomputed, however a good threshold in the number of values needed before recomputing would be desirable to have.

The Class Type Grouping algorithm did not perform as expected. While it did not perform worse than our baseline, it was only marginally better. Filtering out non-relevant classes was expected to clean up our data and produce higher Accuracies. It is believed that a better look at this algorithm is in place.

Lastly, the Weighted Ensemble, while producing the best result in this work, it was not tested with a wide range of parameters. The formulation shown in Equation (3.1) has a weighting parameter α that needs to be chosen. We have set it to 0.5 in this Thesis, after running several tests and noticing that results were satisfactory.

Further studies of this parameter should be carried out. Also, one could decide to not take one of the two terms of the equation if the number of common items is less than a threshold. Preliminary testing of this idea yielded nothing of value, but further testing is needed.

One aspect that was not investigated deeply in this Thesis is the effect of Coverage on Accuracy. Figure (4.1) shows that there is a minimum to the graph. A question that remains open is what would happen if one could recognize difficult candidates, and eliminate them from the global estimation.

This would reduce Coverage, but it is theorized that it might increase Accuracy because of the removal of error prone estimations. We believe this idea could be explored further as it would lead to a Recommender Systems that produces better recommendations, at the expense of breadth.

Bibliography

[1] P. Adamopoulos. *Notes on Recommender Systesm: A Survey of State-of-the-Art Algorithms, Beyond Rating Prediction Accuracy Approaches, and Business Value Perspectives.* Leonard N. Stern School of Business, New York University, (2013).

[2] P. Adamopoulos, A. Tuzhilin. *Recommendation Opportunities: Improving Item Prediction Using Weighted Percentile Methods in Collaborative Filtering Systems.* Proceedings of the 7th ACM conference on Recommender systems, pp. 351-354, (2013).

[3] D. Agarwal, L. Zhang, R. Mazumder. *Modeling Item-Item Similarities for Personalized Recommendations on Yahoo! Front Page.* The Annals of Applied Statistics, Volume 5, Issue 3, (2011).

[4] D. Agarwal, B. C. Chen. *Machine Learning for Large Scale Recommender Systems.* `http://pages.cs.wisc.edu/~beechung/icml11-tutorial/`

[5] Apache Software Foundation. Apache Hadoop. `https://hadoop.apache.org/`

[6] Apache Software Foundation. Apache Mahout. `https://mahout.apache.org/`

[7] R. M. Bell, Y. Koren, C. Volinsky. *All Together Now: A Perspective on the Netflix Prize.* CHANCE, Volume 23, Issue 1, (2010).

[8] R. M. Bell, Y. Koren. *Improved Neighborhood-based Collaborative Filtering.* KDD Cup and Workshop at the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, (2007).

[9] J. Bobadilla, F. Ortega, A. Hernando, J. Alcaln. *Improving collaborative filtering recommender system results and performance using genetic algorithms.* Knowledge-based systems, Volume 24, Issue 8, (2011).

[10] J. S. Breese, D. Heckerman, C. Kadie. *Empirical analysis of predictive algoriths for collaborative filtering.* Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence, pp. 43-52, (1998).

[11] R. Burke. *Hybrid Recommender Systems: Survey and Experiments.* User modeling and user-adapted interaction, Volume 12, Issue 4, (2002).

[12] R. Burke. *Hybrid Web Recommender Systems.* The adaptive web, pp. 377-408, Springer Berlin Heidelberg, (2007).

[13] J. A. Calandrino, A. Kilzer, A. Narayanan, E. W. Felten, V. Shmatikov. *"You Might Also Like:" Privacy Risks of Collaborative Filtering.* Security and Privacy (SP), IEEE Symposium on, pp. 231-246, (2011).

[14] L. Candillier, F. Meyer, M. Boull. *Comparing State-of-the-Art Collaborative Filtering Systems.* Machine Learning and Data Mining in Pattern Recognition, pp. 548-562, Springer Berlin Heidelberg, (2007).

[15] S. H. S. Chee, J. Han, K. Wang. *RecTree: An Efficient Collaborative Filtering Method.* Data Warehousing and Knowledge Discovery, pp. 141-151, Springer Berlin Heidelberg, (2001).

[16] A. S. Das, M. Datar, A. Garg, S. Rajaram. *Google News Personalization: Scalable Online Collaborative Filtering.* Proceedings of the 16th international conference on World Wide Web, pp. 271-280, (2007).

[17] J. Demšar. *Statistical Comparisons of Classifiers over Multiple Data Sets.* The Journal of Machine Learning Research, Volume 7, (2006).

[18] M. D. Ekstrand, J. T. Riedl, J. A. Konstan. *Collaborative Filtering Recommender Systems.* Human-Computer Interaction, Volume 4, Issue 2, (2010).

[19] A. Felfernig, S. Gordea, D. Jannach, E. Teppan, M. Zanker. *A Short Survey of Recommendation Technologies in Travel and Tourism.* OEGAI Journal, Oesterreichische Gesellschaft fuer Artificial Intelligence, Volume 25, Issue 7, (2007).

[20] D. Goldberg, D. Nichols, B. M. Oki, D. Terry. *Using collaborative filtering to weave an information tapestry.* Communications of the ACM, Volume 35, Issue 12, (1992)

[21] G. Guo, J. Zhang, N. Yorke-Smith. *A Novel Bayesian Similarity Measure for Recommender Systems.* International Joint Conferences on Artificial Intelligence (IJCAI/AAAI), (2013).

[22] J. Herlocker, J. A. Konstan, A. Borchers, J. Riedl. *An Algorithmic Framework for Performing Collaborative Filtering.* Proceedings of the 1999 Conference on Research and Development in Information Retrieval, (1999).

[23] J. Herlocker, J. A. Konstan, J. Riedl. *An Empirical Analysis of Design Choices in Neighborhood-Based Collaborative Filtering Algorithms.* Information Retrieval, Volume 5, Issue 4, (2002).

[24] W. Hill, L. Stead, M. Rosenstein, G. Furnas. *Recommending and evaluating choices in a virtual community of use.* Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 194-201, (1995)

[25] Y. Huang. *An Item Based Collaborative Filtering Using Item Clustering Prediction.* ISECS International Colloquium on Computing, Communication, Control, and Management, Volume 4, (2009).

[26] D. Jannach, M. Zanker, A. Felfernig, G. Friedrich. *Recommender Systems, An Introduction.* Cambridge University Press, (2011).

[27] A. Jøsang, R. Ismail, C. Boyd. *A Survey of Trust and Reputation Systems for Online Service Provision.* Decision Support Systems, Emerging Issues in Collaborative Commerce, Volume 43, Issue 2, (2007).

[28] A. Karatzoglou, X. Amatriain, L. Baltrunas, N. Oliver. *Multiverse Recommendation: N-dimensional Tensor Factorization for Context-aware Collaborative Filtering.* Proceedings of the fourth ACM conference on Recommender systems, pp. 79-86,(2010).

[29] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, J. Riedl. *GroupLens: applying collaborative filtering to Usenet news.* Communications of the ACM, Volume 40, Issue 3, (1997).

[30] Y. Koren. *Factor in the Neighbors: Scalable and Accurate Collaborative Filtering.* ACM Transactions on Knowledge Discovery from Data, Volume 4, Issue 1, (2010).

[31] Y. Koren, R. Bell, C. Volinsky. *Matrix Factorization Techniques for Recommender Systems.* Computer, Volume 42, Issue 8, (2009).

[32] Y. Koren. *The BellKor Solution to the Netflix Grand Prize.* Netflix prize documentation, `http://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf`, (2009).

[33] P. D. Kumari, N. S. Reddy. *Survey on Item Based Collaborative Filtering Algorithm for Data Processing and Mining of Patterns in Mobile Commerce.* International Journal of Computer Science and Management Research, Volume 2, Issue 7, (2013).

[34] J. Kunegis, A. Lommatzsch, M. Mehlitz, S. Albayrak. *Assessing the value of unrated items in collaborative filtering*. ICDIM'07, 2nd International Conference on Digital Information Management, pp. 212-216, (2007).

[35] B. Li. *Cross-Domain Collaborative Filtering: A Brief Survey*. 23rd IEEE International Conference on Tools with Artificial Intelligence, (2011).

[36] G. Linden, B. Smith, J. York. *Amazon.com Recommendations, Item-to-Item Collaborative Filtering*. IEEE Internet Computing, Volume 7, Issue 1, (2003).

[37] H. Ma, I. King, M. R. Lyu. *Effective Missing Data Prediction for Collaborative Filtering*. Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 39-46, (2007).

[38] B. Marlin, R. S. Zemel, S. Roweis, M. Slaney. *Collaborative Filtering and the Missing at Random Assumption*. arXiv preprint, Volume 1206, Issue 5267, (2012).

[39] M. R. McLaughlin, J. L. Herlocker. *A Collaborative Filtering Algorithm and Evaluation Metric that Accurately Model the User Experience*. SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, ACM Press, (2004).

[40] *MovieLens 100K dataset* 100,000 ratings (1-5) from 943 users on 1682 movies. `http://grouplens.org/datasets/movielens/`, (1998).

[41] *MyMediaLite: Example Experiments*. `http://mymedialite.net/examples/datasets.html`.

[42] M. O'Connor, J. Herlocker. *Clustering Items for Collaborative Filtering*. Proceedings of the ACM SIGIR workshop on recommender systems. Volume 128, (1999).

[43] M. J. Pazzani, D. Billsus. *Content-Based Recommendation Systems*. The adaptive web, pp. 325-341, Springer Berlin Heidelberg, (2007).

[44] R. Rafter. *Evaluation and Conversation in Collaborative Filtering*. PhD Thesis, University College Dublin, College of Engineering Mathematical and Physical Sciences, (2010).

[45] A. Rajaraman, J. D. Ullman. *Finding Similar Items*. Mining of Massive Datasets, Cambridge University Press, (2011).

[46] N. A. Regi, P. R. Sandra. *A Survey on Recommendation Techniques in E-Commerce*. International Journal of Engineering Research & Technology, Volume 2, Issue 12, (2013).

[47] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, J. Riedl. *GroupLens: an open architecture for collaborative filtering of netnews.* Proceedings of the 1994 ACM conference on Computer supported cooperative work, pp. 175-186, (1994).

[48] F. Ricci, L. Rokach, B. Shapira, P. B. Kantor. *Recommender Systems Handbook.* Springer, (2010).

[49] K. N. Rao. *Application Domain and Functional Classification of Recommender Systems – A Survey.* DESIDOC Journal of Library & Information Technology, Volume 28, Issue 3, (2010).

[50] J. J. Sandvig, B. Mobasher, R. Burke. *A Survey of Collaborative Recommendation and the Robustness of Model-Based Algorithms.* Bulletin of the Technical Committee on Data Engineering, Volume 31, Issue 2, (2008).

[51] B. Sarwar, G. Karypis, J. Konstan, J. Riedl. *Item-Based Collaborative Filtering Recommendation Algorithms.* Proceedings of the 10th international conference on World Wide Web, pp. 285-295, (2001).

[52] J. B. Schafer, D. Frankowski, J. Herlocker, S. Sen. *Collaborative Filtering Recommender Systems.* The adaptive web, pp. 291-324, Springer Berlin Heidelberg, (2007).

[53] S. Schelter, S. Owen. *Collaborative Filtering with Apache Mahout.* Proceedings of ACM RecSys Challenge, (2012).

[54] T. Segaran. *Programming Collective Intelligence - Building Smart Web 2.0 Applications.* O'Reilley, (2007).

[55] U. Shardanand, P. Maes. *Social information filtering: algorithms for automating "word of mouth".* Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 210-217, (1995).

[56] L. Sharma, A. Gera. *A Survey of Recommendation System: Research Challenges.* International Journal of Engineering Trends and Technology (IJETT), Volume 4, Issue 5, (2013).

[57] M. Sharma. *A Survey of Recommender Systems: Approaches and Limitations.* International Journal of Innovations in Engineering and Technology, ICAECE, (2013).

[58] X. Su, T. M. Khoshgoftaar. *A Survey of Collaborative Filtering Techniques.* Hindawi Publishing Corporation, Advances in Artificial Intelligence, (2009).

[59] C. C. Vo, T. Torabi, S. W. Loke. *A Survey of Context-Aware Recommendation Systems*. `http://homepage.cs.latrobe.edu.au/ccvo/papers/16recommendation.pdf`, (2010).

[60] J. Wen, W. Zhou. *An Improved Item-based Collaborative Filtering Algorithm Based on Clustering Method*. Journal of Computational Information Systems, Volume 8, Issue 2, (2012).

[61] G. Xue, C. Lin, Q. Yang, W. Xi, H. Zeng, Y. Yu, Z. Chen. *Scalable Collaborative Filtering Using Cluster-based Smoothing*. Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 114-121, (2005).

[62] Z. Zhao, M. Shang. *User-based Collaborative-Filtering Recommendation Algorithms on Hadoop*. Knowledge Discovery and Data Mining, WKDD'10. Third International Conference on, pp. 478-481, (2010).

Appendix A

Fast Algorithm Alternatives

An inspection of Tables (4.2) to (4.7) for the Default Estimation, and Tables (4.9) to (4.14) for the Class Type Grouping reveals that the Default User Voting and the Indirect Estimation take considerable more time to process than the Plain and the Default Item Voting algorithms.

The reason for this excess Processing Time lays in the way the Top N-Neighbourhood samples are computed. The Default User Voting strategy of the Apache Mahout [6] implementation of the Collaborative Filtering algorithm applies the Default User Voting equations from Section (2.8.2) to the neighbourhood search, and again to the recommendation estimate.

When selecting the neighbourhood, a target needs to be compared for similarity with every candidate in the system. In practice, when using the Plain algorithm the computation is light because the User-Item Ratings matrix is inherently very sparse. However, when applying the Default User Voting or the Indirect Estimation strategies to the neighbourhood selection step, the User-Item Ratings matrix becomes dense, and the number of computations increases significantly.

One possibility to reduce the Processing Time is to avoid applying these strategies during the neighbourhood selection, and only apply them to the recommendation estimate. What this would achieve is a fast neighbourhood selection, with a penalized recommendation estimate, however the estimate computation will generally be done fast because the target there is not compared to every instance, but only to the selected neighbourhood set, which at most is of size N.

The main drawback of using this strategy is that neighbourhood selection step will now be done by using the Plain algorithm, which is deemed inferior to the strategies we put forward. The reason for this claim is that the similarity computation among a target and a candidate is only done for the samples they both have in common, while the other strategies attempt to add the other unrated samples in to the calculation.

We argue that neighbourhood selection is the most important step of a Collaborative Filtering Recommender System, as is the step in charge of finding those samples "similar" to the target, from which to derive the recommendation. If this step is of a lower quality, the recommendation will degrade.

Tables (A.1) to (A.9) show the results of applying the Default User Voting and the Indirect Estimation only to the Recommendation step, but not during the neighbourhood selection. When comparing these results to the ones tabulated in Chapter (4) it can be seen that Accuracy is not as good as the one achieved by using the algorithms also during neighbourhood selection.

Two facts are worth pointing out, Processing Time is now much closer to the Plain algorithm, and the strategies do yield a better MAE, which suggests that both formulation, while to being used for neighbourhood selection, are still superior to the Plain algorithm.

	User Based				Item Based			
	User	User SW	Ind	Ind SW	User	User SW	Ind	Ind SW
MAE	0.7418	0.7478	**0.7390**	0.7457	0.7753	0.7543	0.7745	0.7527
RMS	0.9475	0.9522	0.9446	0.9499	0.9925	0.9652	0.9923	0.9639
Con %	0.9019	0.9536	0.9019	0.9536	0.8551	0.9416	0.8551	0.9416
Time	4.3	4.4	5.1	5.7	4.4	4.8	5.0	6.1

Table A.1: Fast Algorithms, Euclidean Distance similarity, Default Estimation

	User Based				Item Based			
	User	User SW	Ind	Ind SW	User	User SW	Ind	Ind SW
MAE	**0.7877**	0.7941	0.7886	0.7905	0.7941	0.8013	0.8163	0.8017
RMS	0.9931	1.0005	0.9943	0.9952	1.0007	1.0130	1.0211	1.0084
Con %	0.9863	0.9889	0.9863	0.9889	0.9732	0.9847	0.9732	0.9847
Time	4.6	4.8	7.9	8.3	4.7	4.9	6.3	6.8

Table A.2: Fast Algorithms, Pearson Correlation similarity, Default Estimation

	User Based				Item Based			
	User	User SW	Ind	Ind SW	User	User SW	Ind	Ind SW
MAE	0.7892	**0.7880**	0.7891	0.7880	0.9630	0.9394	0.9633	0.9397
RMS	1.0044	1.0029	1.0043	1.0028	1.2212	1.1950	1.2216	1.1953
Con %	0.9712	0.9748	0.9712	0.9748	0.6179	0.6519	0.6179	0.6519
Time	4.7	4.7	7.3	7.5	4.3	4.4	4.5	4.6

Table A.3: Fast Algorithms, Cosine Distance similarity, Default Estimation

	User Based				Item Based			
	User	User SW	Ind	Ind SW	User	User SW	Ind	Ind SW
MAE	0.7482	0.7485	**0.7476**	0.7479	0.7689	0.7601	0.7686	0.7587
RMS	0.9532	0.9523	0.9528	0.9520	0.9901	0.9776	0.9907	0.9766
Con %	0.9325	0.9510	0.9325	0.9510	0.6874	0.8000	0.6874	0.8000
Time	7.5	7.7	7.8	8.1	8.2	8.5	8.7	9.4

Table A.4: Fast Algorithms, Euclidean Distance similarity, Class Type Grouping

	User Based				Item Based			
	User	User SW	Ind	Ind SW	User	User SW	Ind	Ind SW
MAE	0.8069	0.8067	0.8005	0.8001	**0.7909**	0.8055	0.8097	0.8006
RMS	1.0218	1.0216	1.0120	1.0114	1.0072	1.0279	1.0210	1.0135
Con %	0.9732	0.9752	0.9732	0.9752	0.9042	0.9297	0.9042	0.9297
Time	8.4	8.5	9.2	9.2	9.0	8.8	9.5	9.8

Table A.5: Fast Algorithms, Pearson Correlation similarity, Class Type Grouping

	User Based				Item Based			
	User	User SW	Ind	Ind SW	User	User SW	Ind	Ind SW
MAE	0.7942	**0.7940**	0.7942	**0.7940**	0.8468	0.8393	0.8468	0.8393
RMS	1.0083	1.0077	1.0083	1.0077	1.0820	1.0741	1.0820	1.0741
Con %	0.9715	0.9728	0.9715	0.9728	0.6920	0.7120	0.6920	0.7120
Time	8.8	8.9	10.1	9.6	8.8	8.8	9.2	9.2

Table A.6: Fast Algorithms, Cosine Distance similarity, Class Type Grouping

	User Based				Item Based			
	User	User SW	Ind	Ind SW	User	User SW	Ind	Ind SW
MAE	0.7322	0.7374	**0.7305**	0.7361	0.7520	0.7404	0.7515	0.7390
RMS	0.9310	0.9368	0.9294	0.9355	0.9571	0.9441	0.9573	0.9430
Con %	0.8858	0.9315	0.8858	0.9315	0.6535	0.7864	0.6535	0.7864

Table A.7: Fast Algorithms, Euclidean Distance similarity, Weighted Ensemble

	User Based				Item Based			
	User	User SW	Ind	Ind SW	User	User SW	Ind	Ind SW
MAE	0.7899	0.7931	0.7888	0.7900	**0.7780**	0.7898	0.8008	0.7897
RMS	0.9948	0.9983	0.9938	0.9946	0.9800	0.9978	1.0015	0.9932
Con %	0.9704	0.9733	0.9703	0.9733	0.8991	0.9268	0.8991	0.9268

Table A.8: Fast Algorithms, Pearson Correlation similarity, Weighted Ensemble

	User Based				Item Based			
	User	User SW	Ind	Ind SW	User	User SW	Ind	Ind SW
MAE	0.7855	0.7851	0.7854	**0.7850**	0.8860	0.8648	0.8862	0.8650
RMS	0.9969	0.9963	0.9968	0.9962	1.1154	1.0929	1.1157	1.0931
Con %	0.9610	0.9640	0.9610	0.9640	0.5090	0.5421	0.5090	0.5421

Table A.9: Fast Algorithms, Cosine Distance similarity, Weighted Ensemble

Appendix B

Summary of Results

A summary of the results obtained by the different algorithms is tabulated here for convenience, in Tables (B.1) to (B.3). They were used to compute the statistical significance of the algorithms proposed, found in Section (4.5). We include the baseline as computed with the Mahout Collaborative Filtering Recommender System (Plain), and the various algorithms proposed, under User Based and Item Bases approaches, for the three similarity functions studied.

The upper part of the table corresponds to results obtained by the User Based approach, while the lower part of the table corresponds to results obtained by the Item Based approach. Algorithms marked as "FA" refer to the "fast algorithm" variant of the same strategy, in which the neighbourhood selection samples were chosen by using a Plain strategy, and the algorithm in question was only applied to compute the estimation.

For example, the Indirect FA algorithm represents the strategy of using the Plain algorithm during neighbourhood selection, and the Indirect Estimation algorithm to compute the final estimation value. In contrast to the "FA" formulation, the algorithm reported in Chapter (4) employed the Indirect Estimation on both steps, during neighbourhood selection, and during the computation of the estimation value.

	Uniform Weighting		Significant Weighting	
Algorithm	Cov %	MAE	Cov %	MAE
Plain	0.9019	0.7433	0.9536	0.7487
User	0.9422	0.7709	**0.9907**	0.7867
User FA	0.9019	0.7418	0.9538	0.7478
Item	0.9019	0.7429	0.9536	0.7487
Indirect	0.9389	0.7251	0.9850	0.7662
Indirect FA	0.9019	0.7390	0.9538	0.7457
Plain Grouped	0.9325	0.7511	0.9510	0.7510
User Grouped	**0.9497**	0.7551	0.9712	0.7559
User Grouped FA	0.9325	0.7482	0.9510	0.7485
Item Grouped	0.9325	0.7501	0.9510	0.7502
Indirect Grouped	0.9295	0.7278	0.9572	0.7347
Indirect Grouped FA	0.9325	0.7476	0.9510	0.7479
Plain Weighted	0.8858	0.7342	0.9014	0.7325
User Weighted	0.8813	0.7440	0.9014	0.7483
User Weighted FA	0.8858	0.7322	0.9315	0.7374
Item Weighted	0.9019	0.7358	0.9014	**0.7322**
Indirect Weighted	0.9102	**0.7169**	0.9545	0.7421
Indirect Weighted FA	0.8858	0.7305	0.9315	0.7361
Plain	0.8551	0.7870	0.9416	0.7596
User	**0.9584**	0.7373	**0.9985**	0.8185
User FA	0.8551	0.7753	0.9416	0.7543
Item	0.8551	0.7814	0.9416	0.7574
Indirect	0.8493	0.6760	0.8493	**0.6760**
Indirect FA	0.8553	0.7745	0.9416	0.7527
Plain Grouped	0.6874	0.7756	0.8000	0.7615
User Grouped	0.8837	0.7351	0.9883	0.8199
User Grouped FA	0.6874	0.7689	0.8000	0.7601
Item Grouped	0.8000	0.7627	0.6874	0.7738
Indirect Grouped	0.6919	0.6592	0.8663	0.7512
Indirect Grouped FA	0.6874	0.7686	0.8000	0.7587
Plain Weighted	0.6535	0.7612	0.7864	0.7439
User Weighted	0.8786	0.7239	0.9883	0.8134
User Weighted FA	0.6535	0.7520	0.7864	0.7404
Item Weighted	0.6535	0.7579	0.7864	0.7432
Indirect Weighted	0.6543	**0.6395**	0.8659	0.7638
Indirect Weighted FA	0.6535	0.7515	0.7864	0.7390

Table B.1: Summary of Euclidean Distance similarity results

	Uniform Weighting		Significant Weighting	
Algorithm	Cov %	MAE	Cov %	MAE
Plain	0.9863	0.7913	0.9889	0.7916
User	**0.9919**	0.7883	**0.9945**	0.8017
User FA	0.9863	0.7877	0.9889	0.7941
Item	0.9863	0.8005	0.9889	0.7986
Indirect	0.9829	0.7925	0.9907	0.7982
Indirect FA	0.9863	0.7886	0.9889	0.7905
Plain Grouped	0.9732	0.8075	0.9752	0.8068
User Grouped	0.9810	0.7989	0.9834	0.8003
User Grouped FA	0.9732	0.8069	0.9752	0.8067
Item Grouped	0.9732	0.8112	0.9752	0.8097
Indirect Grouped	0.9708	0.7909	0.9743	0.7916
Indirect Grouped FA	0.9732	0.8005	0.9752	0.8001
Plain Weighted	0.9703	0.7928	0.9862	0.7905
User Weighted	0.9855	0.7867	0.9860	0.7982
User Weighted FA	0.9704	0.7899	0.9733	0.7931
Item Weighted	0.9863	0.8000	0.9862	0.7975
Indirect Weighted	0.9668	**0.7848**	0.9730	**0.7881**
Indirect Weighted FA	0.9703	0.7888	0.9733	0.7900
Plain	0.9732	0.8403	0.9847	0.8133
User	**0.9967**	0.7592	**0.9979**	0.8193
User FA	0.9732	0.7941	0.9847	0.8013
Item	0.9732	0.8459	0.9847	0.8174
Indirect	0.9596	0.8274	0.9966	0.8275
Indirect FA	0.9732	0.8163	0.9847	0.8017
Plain Grouped	0.9042	0.8306	0.9297	0.8087
User Grouped	0.9814	0.7559	0.9910	0.8137
User Grouped FA	0.9042	0.7909	0.9297	0.8055
Item Grouped	0.9297	0.8146	0.9042	0.8342
Indirect Grouped	0.9061	0.7649	0.9552	**0.7829**
Indirect Grouped FA	0.9042	0.8097	0.9297	0.8006
Plain Weighted	0.8990	0.8212	0.9731	0.8023
User Weighted	0.9725	**0.7514**	0.9732	0.8109
User Weighted FA	0.8991	0.7780	0.9268	0.7898
Item Weighted	0.9732	0.8336	0.9731	0.8088
Indirect Weighted	0.8944	0.7792	0.9548	0.7942
Indirect Weighted FA	0.8991	0.8008	0.9268	0.7897

Table B.2: Summary of Pearson Correlation similarity results

	Uniform Weighting		Significant Weighting	
Algorithm	Cov %	MAE	Cov %	MAE
Plain	0.9712	0.7891	0.9748	0.7880
User	0.9710	0.8162	**0.9850**	0.8066
User FA	0.9712	0.7892	0.9748	0.7880
Item	0.9712	0.7893	0.9748	0.7881
Indirect	0.9692	0.7994	0.9818	0.7963
Indirect FA	0.9712	0.7891	0.9748	0.7880
Plain Grouped	0.9715	0.7943	0.9728	0.7941
User Grouped	**0.9744**	0.8069	0.9782	0.8057
User Grouped FA	0.9715	0.7942	0.9728	0.7940
Item Grouped	0.9715	0.7944	0.9728	0.7942
Indirect Grouped	0.9707	0.7987	0.9745	0.7983
Indirect Grouped FA	0.9715	0.7942	0.9728	0.7940
Plain Weighted	0.9610	**0.7855**	0.9711	**0.7859**
User Weighted	0.9575	0.8037	0.9670	0.7969
User Weighted FA	0.9610	0.7855	0.9640	0.7851
Item Weighted	0.9712	0.7875	0.9711	0.7860
Indirect Weighted	0.9594	0.7943	0.9696	0.7931
Indirect Weighted FA	0.9610	0.7854	0.9640	0.7850
Plain	0.6179	0.9634	0.6519	0.9397
User	**0.9420**	0.9045	**0.9975**	0.8588
User FA	0.6179	0.9630	0.6519	0.9394
Item	0.6179	0.9633	0.6519	0.9396
Indirect	0.5033	1.0597	0.6527	1.0309
Indirect FA	0.6179	0.9633	0.6519	0.9397
Plain Grouped	0.6920	0.8462	0.7120	**0.8385**
User Grouped	0.9101	0.8775	0.9855	0.8401
User Grouped FA	0.6920	0.8468	0.7120	0.8393
Item Grouped	0.7120	**0.8395**	0.6920	0.8470
Indirect Grouped	0.7228	0.8775	0.7790	0.8731
Indirect Grouped FA	0.6920	0.8468	0.7120	0.8393
Plain Weighted	0.5090	0.8860	0.6174	0.8762
User Weighted	0.6022	0.8868	0.6176	0.8560
User Weighted FA	0.5090	0.8860	0.5421	0.8648
Item Weighted	0.6179	0.8872	0.6174	0.8769
Indirect Weighted	0.4510	0.9744	0.5904	0.9448
Indirect Weighted FA	0.5090	0.8862	0.5421	0.8650

Table B.3: Summary of Cosine Distance similarity results