University of Cincinnati

Date: 2/7/2013

I, Vivek Goyal , hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Science.

It is entitled:

A Recommendation System Based on Multiple Databases.

Student's name: **Vivek Goyal**

This work and its defense approved by:

Committee chair: Raj Bhatnagar, PhD

Committee member: Prabir Bhattacharya, PhD

Committee member: Karen Davis, PhD

3208

A Recommendation System Based on Multiple Databases

A thesis submitted to the

Graduate School

Of the University of Cincinnati

In partial fulfillment of the

requirements for the degree of

Master of Science

in the School of Computing Sciences and Informatics

of the College of Engineering and Applied Sciences

by

Vivek Goyal

B.E. Panjab University, India, 2009

February 2013

Thesis Advisor and Committee Chair: Dr. Raj Bhatnagar

Abstract

Recommendation Systems have long been serving the industry of e-commerce with recommendations pertaining to movies, books, travel packages et cetera. A user's activity or past history of purchases is used to generate predictions for that user. Youtube's video recommendation system, Amazon's "You may also like…" and Pandora's music recommendation system are a few very popular examples. Both explicit and implicit feedbacks are being utilized to churn out predictions about the likings of a customer to recommend items. As recommendation systems have evolved, we primarily encounter two types- Content based and Collaborative Filtering based recommendation systems.

Content based recommendation systems are designed to recommend items similar to the one a user has liked in the past. Recommendation systems based on collaborative filtering recommend items liked by similar users. Users who have liked similar items are identified and items highly liked by those users are recommended. For both content based and collaborative filtering based recommendation systems to predict a rating, it is essential to establish a similarity between items. We have explored correlation and clustering to establish similarity. It was observed that correlation captured similarity better than done by clustering alone. With an intuition that clustering items into similar groups and then employing correlation to determine similarities could improve predictions, we developed an algorithm which is a combination of clustering and correlation that eventually generates prediction for an item rating. We have experimented with adding contextual information to generate better predictions. Our results

suggest that predictions generated by using clustering alone got improved by substituting it with correlation. Further, it was seen that a combination of both improved the predictions over clustering alone but correlation still delivered the best results overall. It was established that bringing in more information may not always help. In this thesis we compare these three algorithms and present our analysis with results.

Acknowledgments

It is by God's grace that I could make it to this day in my career. It has been a long and eventful journey. I am thankful to Dr. Raj Bhatnagar for being my advisor and guiding me wherever I sought advice. His directions and suggestions, invaluable so, have been instrumental in granting me the vision to think apt and eventually making this endeavor of mine successful. I am also grateful to Dr. Prabir Bhattacharya and Dr. Karen Davis for taking out time from their busy schedules and accepting to serve on my thesis committee. I feel specially obliged to Dr. Davis for helping me develop my writing skills very early in my Masters.

I would also like to express my gratitude towards my lab-mates for lending an ear even to my most irrational theories I ever proposed, and for lending a helping hand whenever I went down. I extend my gratitude towards all my friends and my roommate whose support and belief in me has always kept me going.

I would take this opportunity to thank my Uncle Gopal and Aunt Cynthia whose love and care has been unparalleled. They have both eulogized and criticized me at the right place and in the right time which has been effective in getting all my ducks in a row. To plan things ahead of time and never procrastinate are a few things I have learned to practice from them. With their efforts I stand tall today, proud of my achievements. I am also very grateful to uncle Ramesh whose experiences have helped me in my studies to excel. My words aren't enough to thank my cousins who have never shied away from supporting and helping me in whichever way possible. I am deeply thankful to my brother-in-law, Sandeep, whom I pestered for his expert advice even in his odd hours.

Without the encouragement and farsightedness of my parents I would never have been able to embark upon this journey. I would like to express my deepest respect towards them and hope that I stood up to their expectations. My younger brother, Vaibhav, his attitude towards life has inspired me to never worry what went wrong but keep going. I, therefore, dedicate this work to my family.

Contents

Chapter 3: Design of ClusCor Algorithm

Chapter 4: Experiments and Results

Chapter 5: Conclusion and Future Work

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Recommendation Systems

A Recommendation System recommends items to users based on their past preferences and also on other users' past preferences. More precisely, it estimates ratings for those items that a user has not yet rated. It generates predictions for a number of items and presents the items with high predicted ratings as recommendations. Most recommendation systems recommend a set of items to a user. The most visible examples of such systems are online stores that recommend books, movies or other items for sale to their users.

The most widely known and recognized recommendation systems are that of Amazon, Pandora Radio, and Netflix. While Netflix, a provider of on-demand internet streaming media, recommends movies to users they have not rented or rated; Amazon, a multinational online retailer, recommends all sorts of items including books, electronics, clothing etc. Pandora recommends music. "Frequently bought together…" and "Customers who bought this item also bought…" are examples of Amazon's recommendation system at work, as shown in figure 1.1. Recommendation systems also find application in the travel industry to recommend travel packages like Expedia [8], in music industry to recommend favorite music like yahoo music [9] and Pandora radio [12]. Yelp [13] recommends great local businesses like restaurants and coffee places. Facebook, a social networking website, uses a recommendation system to

recommend people who a user might be friends with but does not yet have him/her in his/her "friends list".

Intuitively, estimations/predictions are churned out based on users' past behavior. This behavior can be an *explicit* feedback, where a user puts in effort to indicate likings/dislikings such as ratings for books and movies. It can also be an *implicit* feedback where liking/disliking is inferred from a user's actions like renting a movie or watching a video for comparatively longer period. The latter is termed as "long watch" by the YouTube's recommendation system [11]. In some cases just browsing or searching through is also considered an implicit feedback. It is of interest to note that, although not always, it can be inferred that a user liked one video over another if he/she "long watch"ed it. LinkedIn's "Viewers of this profile also viewed…" is a good example of recommendation based on implicit feedback. YouTube's recommendation system generates predictions by considering a combination of both types of feedbacks [11].

Figure 1.1: Amazon's recommendation system in action [7].

Not only are recommendation systems used in the field of e-commerce, law enforcement and intelligence agencies can use recommendation systems that recommends top-k potential suspects of a crime [20].

1.2 Types of Recommendation Systems

As more and more research is being done in the field of Recommendation Systems and Information Retrieval, recommendation systems are seen to have evolved over a period of time. Various different Machine Learning techniques and Approximation Theory methods are generally employed on the feedback from users to generate predictions. Based on their approach to estimate ratings, a Recommendation System can broadly be classified into three categories: Content-based, Collaborative Filtering based, and Knowledge-based Recommendation Systems [15].

1.2.1 Content-based Recommendation Systems estimate ratings based on the contents of items that a user rated highly or has shown interest in the past. It looks for those items in the system that the user has not yet rated and those which are most similar in content to the items the user has highly rated. These recommendation systems are based on the conjecture that if a user has liked a particular item, he or she is likely to also like the ones most similar to it. For example, in a movie rating scenario, if a user has highly rated the movie *The Matrix*, he/she might also like *Star Trek* since both of these are of the same genre: Action, Adventure and Sci-Fi. Pandora, an online music recommendation system, is a good example of content-based recommendation systems. It recommends music similar to what users have liked in the past. However, recommendation systems based on this concept are paralyzed by the problem of

over-specialization [4]. This means that a user who has rated only romantic movies will never receive recommendations for even the best horror movies.

1.2.2 Collaborative Filtering Recommendation Systems recommend items highly rated by those users who have similar tastes rather than recommending similar items. This is very much similar to being guided by public opinion. The concept is based on the premise that a user is likely to have interests in items that are liked by those who have similar tastes. For example, again in the movie rating scenario, suppose we want to recommend a movie to a user, say Bob. Suppose there are other users like, Jay and Alice who have all liked, as Bob has too, the movies *The Hills Have Eyes*, *House of Wax*, *Texas Chainsaw Massacre* and *Paranormal Activity*. Then a movie highly rated by these users but not yet rated by Bob can be a candidate for recommendation to Bob. This way a movie, say *If Only*, which is of a completely different genre, can be a recommendation for Bob. This solves the problem of over-specialization (when movies are recommended based on the ones already rated by a user).

1.2.3 Knowledge-based Recommendation Systems utilize knowledge acquired by the users to recommend items. For example, the entrée restaurant recommender [14], recommended restaurants similar to the ones a user indicated to have liked at a different location. It first acquires knowledge by asking the user to specify what's desired such as type of cuisine, price, style, atmosphere etc. Then based on the closest match to this acquired knowledge it would recommend a local restaurant [14].

Approaches to recommendation systems can also be categorized as personalized recommendations, which are based on users' past behavior; social recommendations, which

are based on past behavior of other similar users; and item recommendations based on items of interest [16].

From among various different ways of capturing the notion of similarity, in this thesis we examine clustering, correlation and a combination of both. We compare and contrast them for making recommendations. Clustering is a common technique to determine similarities in a given dataset. From the output of clustering, one can visualize the data points as belonging to different groups and thus establish similarity if two points are in the same cluster or dissimilarity if they fall in different clusters. Pearson's correlation [2] captures a different notion of similarity and significantly differs from what clustering depicts. Given a set of data points, consider for example a set of movie ratings, correlation is calculated between ratings of two movies over the complete data set. This provides an insight on the behavior observed in the ratings of those two movies by different users. If the correlation is positive it indicates that the ratings given to the two movies are very similar. This means that if one movie is rated high, it is very likely that the other movie will also be rated high (and vice versa) by majority of the users. If, however, there is a negative correlation it means that if one of the movies is rated high, the other is seen to be rated low by most of the users. Zero indicates no correlation at all. Most of the existing recommendation systems use one of these two ways to determine similarity between items.

Our proposed approach to determine the predictions seeks to combine both the above approaches of determining similarity. That is, we first cluster the items, i.e. movies, based on their genre information. The items are grouped using a fuzzy clustering algorithm, keeping in mind that one item can belong to multiple groups at the same time. Hence, each item is associated to a group/cluster with a certain degree of membership. Once we have the items clustered, we calculate correlation with only those movies which are most similar to the one for

which a prediction has to be calculated. The intuitive idea behind this approach is that if we seek to determine the rating of a movie for a user then instead of basing it on all other movies that have similar ratings, we should base it only on those other movies that have similar ratings and are also on the same fuzzy genre. That is, a user's preference for a movie from a fuzzy genre depends on his ratings for other movies of the same fuzzy genre. This, we expected, would give us better ratings and also reduce the computational load of the recommendation algorithm.

1.3 Our objective

As explained above, Collaborative Filtering technique tries to find users with similar tastes and recommends items highly rated by these users. A number of similarity measures have been suggested so far to estimate these similarities. Pearson's correlation is one of the effective measures proven to generate good estimates which eventually lead to a better prediction [2].

1.3.1 Our Contribution and Approach

In this thesis we have explored the problem of integrating knowledge from two different databases to bear upon the generated recommendations. For example, let us say we have a database of user ratings of movies and another database of movies and the genres to which they belong, figure 1.2.

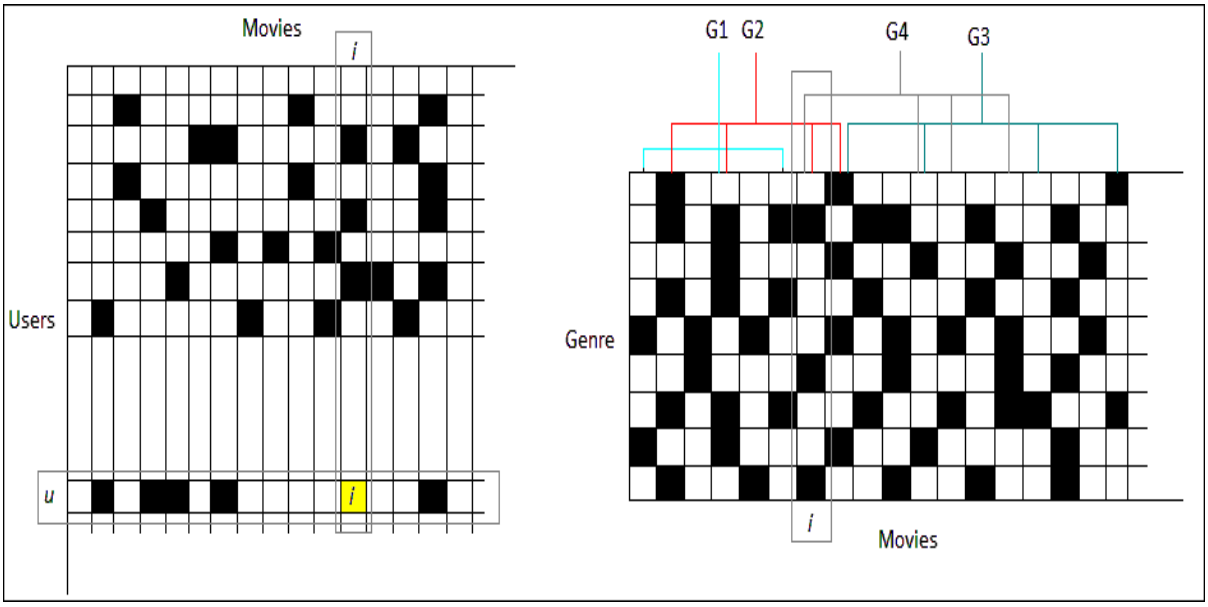


Figure 1.2: Movie ratings matrix and movie genre matrix. Solid block indicates value recorded.

The matrix on the left shows ratings given by a user to a movie. A solid block in figure indicates that a rating has been recorded. Similarly, the matrix on the right holds movies and displays which genre each movie belongs to. We first make overlapping cluster/groups G1, G2 and so on of movies as shown in this figure. Note that one movie can belong to multiple genre groups at once. This is achieved by performing fuzzy clustering. Now, suppose in the matrix on the left, we are interested in predicting a rating for the movie *i* by user *u*. The corresponding block is shown in yellow ink. From the matrix on the right, in a set S1 we pick all the movies that belong to the same cluster as *i*. In the figure, *i* belongs to groups G2 and G4. In such a case we choose the cluster to which it belongs with a highest degree of membership. We then take the intersection of set S1 and the set of movies user *u* has already rated. Let's call this set S2. We calculate similarity values using Pearson's correlation for each of the movies in the set S2 with the movie *i* for which a prediction is to be generated. The predicted rating is then calculated by taking the weighted average (similarity being the weight) of the ratings given to top 20 movies in set S2 which are most similar to *i*. At times the intersection yields less than 20 movies. In such cases we take all movies into count.

In this thesis we have developed and tested our proposed and other algorithms in the context of the *Movielens* dataset which is publicly available for research purposes [6].

Traditionally, a user's potential rating for a movie will be determined based upon the ratings of all other movies by those users whose ratings have a strong correlation with the movie in question. But our hypothesis is that we should restrict the computation of the potential rating to only those movies that are in some way similar to the movie to be rated. We go to the second database to form fuzzy clusters of movies according to the genres to which they belong. Then in the first database we select only those movies for computing a potential rating that belong to the same fuzzy cluster in the second database and also have a strong correlation with the movie to be rated.

We have used fuzzy clustering of movies in the genre space to create overlapping clusters of movies. This is also because a movie belongs to multiple genres and the prototype of a fuzzy cluster consists of multiple genres, each genre with a different membership value. In the first database we have used empirical correlation, based on Pearson's correlation, to determine the movies that are strongly correlated with the movie to be rated.

We have presented our analysis of the above approach and shown the results. Our analysis shows that for some cases our approach provides better prediction but for some other cases the prediction worsens. Our proposed approach is compared to two other baseline approaches: clustering alone and Pearson's correlation alone for making the predictions. With plots drawn to examine and compare the performance of the three different ways to predict a rating, it is evident that the algorithm we have proposed and developed falls in between the other two. But it does not consistently improve beyond the performance of the other two algorithms. Our

algorithm outshines when compared to the one which generates similarities based on clustering alone. It makes better predictions in approximately 40% of the cases than the one which uses correlation alone as a criterion to calculate similarities. The plots show a comparison between our algorithm and the one that implements neighborhood models based on Pearson's correlation. The intuitive result that we can draw from our tests and implementations is that in the case of movies, the genre information of a movie to be rated may not always help in improving a user's potential rating for it.

Chapter 2

Related Work

In this chapter we present the other research work related to our thesis work that has been done in the past. We discuss significant work done in two different research areas and show how they are related to our proposed approach and how it is different from what we have accomplished. We begin with discussing the neighborhood models based on Collaborative Filtering techniques [2]. We then discuss the Multidimensional Recommendation Model proposed by Adomavicius et al. [19].

2.1 Neighborhood Models Based on Collaborative Filtering

A common approach to generate recommendations is Collaborative Filtering. And, the most common approach to Collaborative Filtering techniques is based on neighborhood models. The one that has become more popular is item-based approach as it offers scalability and improved accuracy over the user-based approach [2]. In a neighborhood model we basically classify the unknown entity by considering certain number (*k*) of its nearest neighbors. The simplest way to make a decision is typically to go with the majority. The entity is said to be inclined towards taking the properties of the ones in majority in the pool of *k* nearest neighbors. Let us take a quick look at how user-based and item-based approaches to Collaborative Filtering work.

2.1.1 User-based Collaborative Filtering: To predict a rating for an item *i* by a user *u*, the ratings by those users who are most similar to *u* are considered. Similar users are the ones

who have rated items rated by *u*, very similarly. Hence, to predict an unknown rating, those users are selected who are similar to *u* in their other ratings and have also rated *i*. An average or weighted average over the ratings for *i* by these similar users can then be taken as the predicted rating for *i* by user *u*. As is generally the case with Collaborative Filtering, the predictions would be more accurate when the users have rated substantial number of items in the dataset.

	Item1	Item2	Item3	Item4	Item5
User1	1	-	5	4	?
User2	4	2	2	3	
User3	2	4	4	3	4
User4	1	4	-	3	5
User5	2	5	4	5	5

Figure 2.1: Example dataset to demonstrate User-based Collaborative Filtering [7]

Figure 2.1, shows an example dataset where rows represent users and columns represent items. The values indicate ratings between 1 and 5, 1 being poor and 5 being exceptionally good. A rating is to be predicted that User1 might give to Item5. First, in a set, we select all those users who have rated Item5. This set will have {User3, User4, User5}. From this set we keep only those who have also rated the items rated by User1. Thus the set will now have {User3, User5} Using appropriate similarity measure we see that these users have rated the items common to those of User1 similarly- Item1 is rated low by all three, Item3 & Item4 are highly rated. Hence we say that User3 and User5 are User1's neighbors. Suppose the similarity between User1 & User3 is denoted by S_{13} and that between User1 & User5 by S_{15}. Also suppose that the similarity values come out to be $S_{13} = 0.7$ and $S_{15} = 0.6$. Then the predicted rating is calculated as

$$\frac{4*0.7 + 5*0.6}{0.7 + 0.6} = 4.46$$

2.1.2 Item-based Collaborative Filtering: To predict a rating for an item *i* by user *u*, the ratings for items rated by *u* and most similar to *i* are considered. An average or weighted average of these ratings can be taken as the predicted rating for the item. The idea central to the algorithm is a similarity measure which gives an estimate on how similar any given two items or users are, as is the case with user-based Collaborative Filtering. Rating is predicted by taking the weighted average of the ratings of neighboring items, similarity being the weights. The content-based recommender system tries to understand the commonalities among the items the user has rated highly in the past. Then, only the items that have a high degree of similarity (specific actors, directors, genres, authors, subject matter etc.) to whatever user's preferences are would be recommended [4].

	Item1	Item2	Item3	Item4	Item5
User1	3	-	2	5	?
User2	2	2	-	4	
User3	3	2	3	5	4
User4	4	4	1	4	5
User5	4	4	1	4	4

Figure 2.2: Example dataset to demonstrate Item-based Collaborative Filtering [7]

Figure 2.2 shows an example dataset similar to that in figure 2.1 above. Suppose we want to predict a rating User1 could possibly give to Item5. Here we wish to find items rated by User1 that are rated most similar to Item5 over the rest of the complete dataset. That is, we wish to find out which of the items- Item1, Item3 and Item4 are rated most similar to Item5. To estimate the similarity between Item4 and Item5 (S_{45}), we select all the users who have rated both of these. Suppose using an appropriate similarity measure, most commonly Pearson's Correlation,

we estimate the similarity S_{45} to be 0.85. In the same fashion suppose that the estimates for the similarities S_{35} and S_{15} are -0.33 and 0.78, respectively. Since a negative similarity indicates that if one item was rated high the other was rated low over the complete dataset, we strike out Item3 from the set of items rated most similarly to Item5. This leaves only Item1 and Item4 in the set with similarity values S_{15}=0.78 and S_{45}=0.85. The predicted rating for Item5 by User1 will be calculated as-

$$\frac{4*0.78 + 5*0.85}{0.78 + 0.85} = 4.52$$

Hence, if the dataset is visualized as an *m-by-n* matrix where rows are users and columns are items, then similarities in case of user-based Collaborative Filtering would be computed along the rows but it is computed along the columns in case of Item-based Collaborative Filtering.

2.1.3 The Neighborhood Algorithm

Algorithm *correlation*, as listed in figure 2.3, predicts a rating for item *i* by the user *u*. It starts with generating an array of items the user has rated so far. This array is denoted by I(*u*) as shown in line-1 of the algorithm 1 in figure 2.3. Depending upon the dataset used, the size of this array can be anywhere from a few entries to a few hundred entries for a typical user. The *MovieLens* dataset is cleaned to have every user rated at least 20 items. Hence, this array has at least 20 elements.

For each item *j* in I(*u*), a similarity with the item *i* is then calculated in lines 2-4 and stored in another array denoted by S. The length of this array has to be the same as that of I(*u*). The algorithm used to calculate these similarities, similarity (*i*,*j*), is discussed later in this section. Similarity between item at the index *n* of I(*u*) and the item *i* is stored at the index *n* of the array *S*.

Algorithm 1: correlation(u,i**)**

Input: String- user *u*, item *i*

Output: float- predicted rating, r_{ui} //value between 1 & 5

1: array I(*u*) ← all items rated by user *u*;

2: for each item *j* ϵ I(*u*) **do**

3: **array** S ← **similarity**(*i*,*j*);

4: end for

5: array $S^k(i,u)$ ← top K items rated by *u* most similar to *I*;

6: for each item *j* ϵ $S^k(i,u)$ **do**

7: numerator += S_{ij} * (r_{uj} - b_{uj}) ;

8: denominator += S_{ij};

9: end for

10: r_{ui} = b_{ui} + (numerator / denominator);

11: return (r_{ui});

Figure 2.3: Algorithm to predict rating based on the item-item approach of Collaborative Filtering (CF)

Next, in line 5, the elements in array I(*u*) are sorted on similarity values from S and, top *k* items are copied in a yet another array denoted by $S^k(i,u)$. This is a set of *k* items, most similar to the item *i,* that user *u* has rated. These *k* items are the *k* nearest neighbors of item *i* and the metric to measure closeness is “similarity”. Only these *k* nearest neighbors are considered to predict a rating for the item *i*.

Predicted value of the rating is taken as a weighted average of the ratings of neighboring items, while adjusting for user and item effects through the baseline predictors. This essentially means, rather than using the ratings as is, they are adjusted to add/subtract a value which is a measure of how a particular user is biased to rate a particular item. For example, in Algorithm1 b_{ui} represents the value by which user *u* is biased to rate the item *i* more or less than the average rating over the complete dataset. This value is estimated by looking at the user's and item's rating patterns from past and is discussed later in this section.

Since the calculation of similarities between two items involves the concept of baseline predictors, it is vital to discuss these here. Baseline predictors capture the deviation in rating that a user is likely to make for a given item from the average rating over the complete dataset. Also known as bias, it is denoted by b_{ui} and can be expressed as:

$$b_{ui} = \mu + b_u + b_i$$

Here, μ is the average rating on the complete set. The parameters b_u and b_i indicate the observed deviations of user *u* and item *i*, respectively, from the average [2]. For example, suppose in a typical dataset we want a baseline predictor for the rating of the movie Titanic. Suppose the average rating over all the movies is 3.7 out of 5 stars. Furthermore, Titanic is better than an average movie, so it is *biased* to be rated 0.5 stars above the average. On the other hand, Bob is a critical user who is *biased* to rate 0.3 stars lower than the average. Thus the baseline predictor in this case would be 3.9 by calculating 3.7 + 0.5 + (-0.3). There are more than one ways to calculate user and item biases [2].

2.2 Similarity Measures Used in Collaborative Filtering Technique

It is of importance to understand that collaborative filtering differs from content based techniques in the way that while the latter seeks to find items similar to those rated highly by the same user, the former looks for users with similar tastes and recommends what they liked most. Depending on the type of data and nature of prediction, one of the following can be used [3].

2.2.1 Pearson's Correlation Coefficient

One of the most commonly used similarity measures, Pearson's correlation coefficient, is a linear correlation between two vectors representing ratings on the same set of items. Consider the table shown in figure 2.4 below.

	Item i	Item j
User12	3	4
User24	2	3
User35	4	5
User43	3	4
User76	4	5

Vector 1 = {3,2,4,3,4}
Vector 2 = {4,3,5,4,5}

Figure 2.4: Sample set of users who co-rated items *i* and *j*.

It correctly suggests whether there is an increasing or decreasing trend shown by a variable if an increase in the other is noticed. The values range from -1 to 1. -1 indicating a perfectly negative correlation which means that the variable under inspection decreases in proportion with an increase in the other; 1 indicating a perfectly positive correlation while 0 meaning no correlation at all. Denoted by ρ, mathematically it can be expressed as:

$$\rho_{ij} = \frac{\Sigma_{u \epsilon U(i,j)} \left(r_{ui} - \overline{r_i} \right)\left(r_{uj} - \overline{r_j} \right)}{\sqrt{\Sigma_{u \epsilon U(i,j)} \left(r_{ui} - \overline{r_i} \right)^2 \cdot \Sigma_{u \epsilon U(i,j)} \left(r_{uj} - \overline{r_j} \right)^2}}$$

Here, the set U(i,j) is the set of users who have rated both items *i* and *j*. The observed rating for item *i* by user *u* is denoted by r_{ui}. $\bar{r}_i$ is the average rating for ratings given to item *i* by all the users who co-rated *i* and *j*. Hence, it will be the average of Vector 1 shown in figure 2.4. Similar holds for Vector 2.

Pearson's correlation uses the *mean* of the rating vectors of co-rated items by both users. Since the rating matrix is generally very sparse, indicating that not many users have rated many items, this *mean* would not be a good criterion. Hence we use empirical correlation coefficient instead, wherein we replace the mean with baseline predictors detailed earlier. The empirical correlation denoted by *ρ'* can thus be expressed as:

$$\rho'_{ij} = \frac{\Sigma_{u \epsilon U(i,j)} (r_{ui} - b_{ui})(r_{uj} - b_{uj})}{\sqrt{\Sigma_{u \epsilon U(i,j)} (r_{ui} - b_{ui})^2 \cdot \Sigma_{u \epsilon U(i,j)} (r_{uj} - b_{uj})^2}}$$

Where b_{ui} denotes the bias with which user *u* rates the item *i*, and r_{ui} is the rating for item *i* by user *u.* The similarity can further be calculated as

$$S_{ij} = \frac{\eta - 1}{\eta - 1 + \lambda} \rho'_{ij}$$

Where η = |U(i,j)| i.e. number of users who co-rated items *i* and *j*. A typical value of λ is 100 [2].

2.2.2 Cosine Similarity

The concept of cosine similarity is taken from Information Retrieval. It measures the cosine of angle between two vectors where a smaller angle indicates greater similarity. In information retrieval it is used to estimate similarity between two documents thus suggesting whether to

classify them under the same class or different classes. The value ranges from -1 meaning exactly opposite, to 1 which means exactly the same. A value of 0 indicates independence. For the example shown in figure 2.4, cosine similarity can mathematically be expressed as:

$$S_{ij} = \frac{\sum_{u \epsilon U(i,j)} (r_{ui} \cdot r_{uj})}{\sqrt{\sum_{u \epsilon U(i,j)} (r_{ui})^2} \cdot \sqrt{\sum_{u \epsilon U(i,j)} (r_{uj})^2}}$$

Where U(i,j) is a set of users who co-rated items *i* and *j*. r_{ui} and r_{uj} represent observed ratings by the user *u* from the set U(i,j) for items *i* and *j*, respectively.

2.2.3 Adjusted Cosine

Cosine similarity, as explained above, does not take into account the difference in each user's rating scales. This drawback is addressed in what is known as Adjusted-Cosine by subtracting the corresponding average rating from each of the vector elements.

$$S_{ij} = \frac{\sum_{u \epsilon U(i,j)} (r_{ui} - \overline{r_i}) \cdot (r_{uj} - \overline{r_j})}{\sqrt{\sum_{u \epsilon U(i,j)} (r_{ui} - \overline{r_i})^2} \cdot \sqrt{\sum_{u \epsilon U(i,j)} (r_{uj} - \overline{r_j})^2}}$$

$\overline{r_i}$ is the average of ratings by all users in the set U(i,j) given to item *i*. Similarly, $\overline{r_j}$ is average rating given to item *j* by users in the set.

2.2.4 Clustering

Clustering has been widely used to classify similar entities under one group or cluster. This way all the similar items get grouped together forming several groups/clusters having relatively different characteristics. But clustering by itself is not a very efficient way of estimating similarities as it does not indicate on a scale how similar two items are to each other. Depending on the nature of data and the clustering characteristics desired, there are different algorithms to perform clustering. The most commonly employed would be K-means, Hierarchical/Agglomerative and Density-based clustering. Each of these is an example of *hard clustering* which means that each element in the dataset can belong to at most one cluster, possibly leaving some of the elements belonging to none. These elements are called outliers. It is not always desirable to "hard cluster" items. For this reason, there are algorithms that would allow an element to belong to more than one cluster, each with a certain degree of membership. The algorithms based on this concept are known as *soft* or *fuzzy clustering* algorithms.

For the purpose of this research we have used the empirical correlation coefficient to calculate item similarities. The algorithm to calculate similarity between two items is listed in figure 2.5. The similarity measure described here is based on Pearson's Correlation coefficient ρ_{ij}, which measures the tendency of users to rate items *i* and *j* similarly [2]. Pearson's correlation uses mean values in its calculations. With most of the fields in the user-item matrix being empty, the mean will not be a good estimation. Hence, we instead use empirical correlation which replaces the mean values with baseline predictors.

Algorithm 2: similarity(i,j)

Input: item *i*, *j*

Output: float- similarity, S_{ij}

1: array U(*i*, *j*) ← users who co-rated *i* & *j*;

2: for each user *u* ϵ U(*i*,*j*) **do**

3: numerator += $(r_{ui} – b_{ui})*(r_{uj} - b_{uj})$;

4: denominator1 += $(r_{ui} - b_{ui})^2$;

5: denominator2 += $(r_{uj} - b_{uj})^2$;

6: end for

7: denominator = $(\text{denominator1} + \text{denominator2})^{1/2}$;

8: ρ_{ij} = numerator / denominator;

9: η = |U(*i, j*)|;

10: $S_{ij} = [(\eta – 1) / (\eta – 1 + \lambda)] * \rho_{ij}$;

11: return (S_{ij});

Figure 2.5: Algorithm to calculate similarity, using correlation, between items *i* and *j*.

Algorithm, as shown in figure 2.5, starts with generating an array/set, U(*i*, *j*) of users who rated both items *i* and *j*. With these users the algorithm tries to find out if there is a correlation as to how similarly the two items tend to be rated. The value of this correlation, ρ_{ij}, lies in the range of -1 to 1. Here, negative correlation indicates that if one item was rated high, the other was rated low or the other way around. Similarity value, denoted by S_{ij}, is a shrunk correlation coefficient

of the form as in line 10 of the algorithm. The variable η denotes the number of users who rated both items. A typical value of λ is 100 [2].

Note again that the similarity value here is not a measure of how similar the two items are content-wise. But it tells how similarly do users tend to rate the items. That is to say that if a user has highly rated one of the items, how likely is the user to rate the other item also high.

Figure 3.3 in the next chapter shows the complete algorithm we propose to investigate if the predictions generated by the neighborhood model, as depicted in the figure 2.3, can be improved by incorporating contextual information about the items. We call this algorithm ClusCor.

The time complexity of this algorithm depends on the set of users who co-rated 2 items between which similarity is to be determined. In the worst case, this set can only be as big as the set of number of users, given by *N.* Hence the time complexity of this algorithm would be O(N).

2.3 Multidimensional Recommendation Model

Adomavicius et al. proposed to incorporate contextual information into the then existing recommendation system models to improve accuracy and relevancy of recommendations [19]. Their theory was primarily based on the hypothesis that in many applications it may not be sufficient to only consider 2 dimensions viz. users and items to produce recommendations. Incorporating contextual information of user's decision scenario was taken to be of immense importance. For instance, in the case of a recommendation system that recommends vacation packages, not only are the user and item important but incorporating the time of the year would make the recommendations more relevant. As an example, summer season is generally

considered to be the best time of the year to visit Niagara Falls. Hence, it will be relevant to recommend a couple to visit Niagara Falls in summers rather than in the winter season.

To be able to provide such recommendations, Adomavicius et al. proposed a multidimensional recommendation model that would include more dimensions than just user and item to generate relevant recommendations. This can be visualized as shown in figure 2.6 that shows a 3D model.

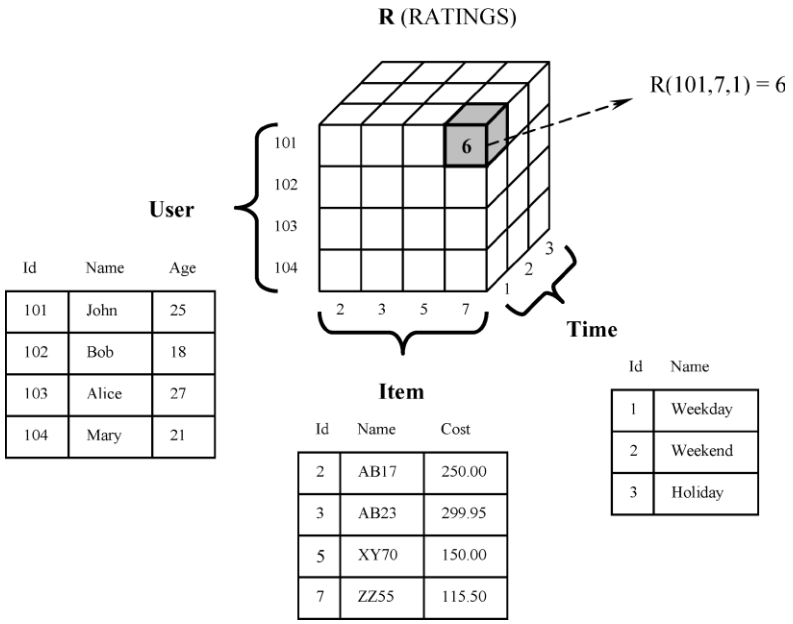


Figure 2.6: Multidimensional model proposed by Adomavicius et al. [19]

In figure 2.6 the multidimensional cube shown, referred to as ratings cube, depicts a 3 dimensional model of a recommendation system where each cell represents a rating of an *item* given by a *user* at a particular *time*. The highlighted cube, for example, suggests that user with ID *101* has given a rating of magnitude 6 to the item with ID *7* at time *1*. Hence there will be 3 corresponding tables or records that store information about each of the 3 dimensions, as shown in figure 2.6.

It is important to note that this model supports storage of measurements in the multidimensional cubes. The rating value of 6 as highlighted in figure 2.6 is an example. One of the key characteristics Adomavicius et al. pointed out was the capability of this model to aggregate these measurements at different levels of hierarchy [19]. For instance, in the movie rating scenario, all the items (movies) can be grouped into sub categories based on genre information. The users can similarly be grouped based on their demographic information like age or marital status or occupation etc. The capability to aggregate the measurements leads to better and more relevant predictions. For example, now not only we know how a user likes a particular movie, but with aggregating movies based on their genre information we can find out how the user likes a particular genre. On the same lines we can also find out how a particular group of users, say engineers, like a particular genre.

In our work we have tried to aggregate items (movies) based on their genre information and perform collaborative filtering on top to generate predictions. The idea is to restrict the computation of the potential rating to only those movies that are in some way similar to the movie to be rated.

The way in which it differs from our work is that where Adomavicius et al. proposed to introduce contextual information as a third dimension. We, on the other hand, have used the contextual information about movies, i.e. genre, to cluster the items first and reflect this grouping of items on the ratings dataset. This way we have avoided inclusion of the irrelevant items to be included in making a prediction for another item.

Chapter 3

Design of the ClusCor Algorithm

We, in this research, have designed an algorithm that can be conceived as a black box that takes as input the user *u* and item *i*. Using the contextual information about the items in the dataset it predicts a rating user is likely to give to an item. The algorithm is based on collaborative filtering technique and is an improvement over the neighborhood models proposed by Koren et. al. [2]. Figure 3.1 shows how ClusCor algorithm works. We discuss each of the aspects depicted here in this chapter with a detailed explanation of the algorithms that drive them.

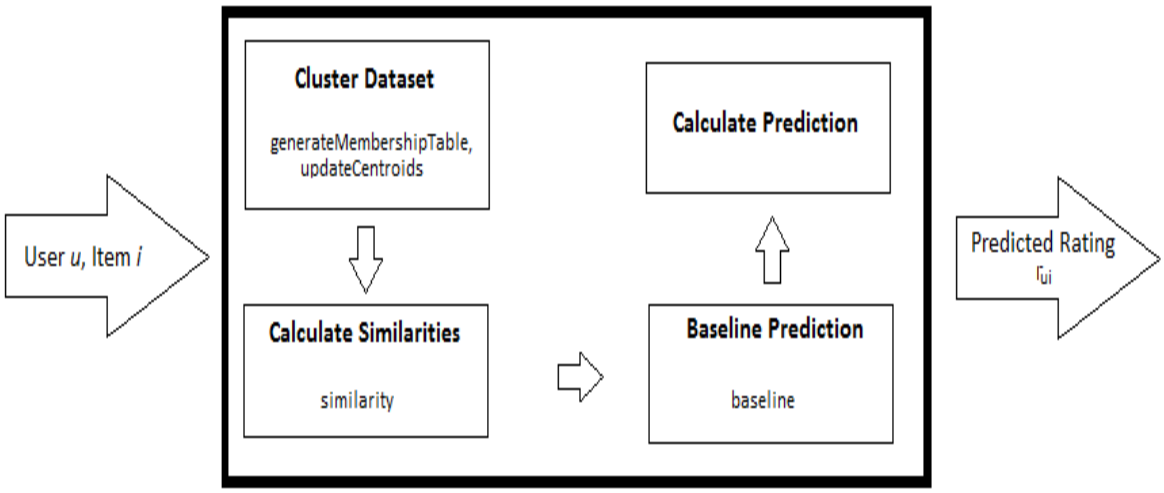


Figure 3.1: The ClusCor algorithm

The name ClusCor comes from CLUStering and CORrelation. These are the two key concepts that drive this algorithm. Abstractly, the algorithm clusters items in the dataset based on the contextual information about them. It then picks and uses only the relevant items to calculate the similarities using correlation and ultimately generates a prediction. Clustering helps pick only the relevant items to calculate the similarity. This way we manage to circumvent the contributions by items which should not have a say in deciding the similarities. For example, consider a hypothetical user in a movie rating scenario. Suppose that the user has rated a set of movies as indicated in the table shown in figure 3.2. It is just an example and it should be understood that the actual scenario could be a lot bigger than what is presented here.

Movie id	1	2	3	4	5	6	7	8	9	10
Rating	4	4	5	4	4	4	3	4	2	3

Figure 3.2: Table of ratings (1 being poorest, 5 being strongest) given by user *u* to different movies.

Let us assume that the movie ids from 1 to 5 fall under a certain category of genre, say category A, and the rest fall under a different category of genre, B. This essentially says that a movie from one category is not similar in content to any movie from a different genre category. Suppose we want to predict a rating this user is likely to give to a movie with id 11 that falls in the set A of movie categories. The correlation algorithm as depicted by figure 2.3 tries to calculate the similarity between a movie, one by one, from the table and the movie for which the prediction has to be made. This value of similarity between any given two movies is calculated by considering the ratings given to both the movies by other users in the dataset. It is determined based on the similarity values as to which of the movies rated by this user are similar to the one we need to predict a rating for. A weighted average of the ratings of the *k* most similar movies is then taken as the predicted rating. A drawback here is that in the set of *k* most similar movies there can easily be the ones that come from different category of movies and hence can potentially harm the prediction by pushing it either side.

The ClusCor algorithm tries to circumvent those irrelevant movies in the above example to present a better and meaningful prediction. We explain the algorithm in detail in section 3.2.

3.1 The Dataset

The dataset used in this thesis is a movie ratings dataset, *MovieLens-100k*. It is collected and publicly made available for research by the research group GroupLens Research [5] at the Computer Science department of University of Minnesota [1]. It is a collection of 100,000 ratings by 943 users on a total of 1682 movies. It can be seen as a 943x1682 matrix with ratings as field values. This data was collected through the MovieLens website during the seven-month period from September 19th, 1997 through April 22nd, 1998 [6]. Movies and users each have unique ids to distinguish them from others. The ratings are strictly integers from 1 to 5 inclusive, 1 representing the weakest and 5 the strongest rating.

3.2 The Algorithm

The ClusCor algorithm begins by clustering the items in the dataset based on the contextual information which in our case of movie ratings is genre (line 1 in figure 3.3). Rather than having a movie exclusively assigned to only one cluster, we propose a degree of membership (DOM) with which a movie belongs to a particular cluster. This is based on the fact that a movie can possibly belong to more than one genre at the same time. The center or prototype of each cluster is also a set of genres, with a membership value attached to each genre. By the time fuzzy clustering is done, we have a membership table that holds the information as to which movie has what DOM values for each of the clusters. This is known as fuzzy or soft clustering since no item strictly belongs to only one cluster. Note that for any item, the sum total of all the DOM values will be unity. A DOM value closer to 1 depicts a higher association of that item with the given cluster. The complete clustering algorithm is shown in figure 3.3.

Algorithm 3: ClusCor(u, i)

Input: user *u*, item *i*, dataset.

Output: float- rating.

1: **clusterData**(dataset);

2: clusterId = **whichCluster**(*i*);

3: array clusterTopN ← **getTopN**(clusterId,N);

4: array I ← items rated by user *u*;

5: array selectedItems ← clusterTopN ∩ I;

6: **for each** j ϵ selectedItems **do**

7: **if**(**similarity**(i,j) > 0) **then**

8: similarItems.add(**similarity**(i,j));

9: **end if**

10: **end for**

11: $S^k(i,u)$ ← top k items from the sorted array similarItems;

12: **for each** j ϵ $S^k(i,u)$ **do**

13: b'_{uj} = **baseline**(u,j); //b' is adjusted baseline predictor.

14: numerator += S_{ij} * (r_{uj} – b'_{uj});

15: denominator += S_{ij};

16: **end for**

17: r'_{ui} = b'_{ui} + (numerator / denominator);

18: **return**(r'_{ui});

Figure 3.3: ClusCor: Collaborative filtering algorithm to predict ratings using contextual information about items.

Now, suppose we want to generate a prediction of the rating that the user *u* is likely to give to the item *i*. In line 1 of the algorithm, the items are clustered and a membership table is also

generated. The next step, line 2, is to find the cluster to which the item *i* belongs with the highest DOM value. Next, in line 3, it fetches the top N items that belong to this cluster with the highest degree of membership and puts them in an array named *clusterTopN*. In a different array, *I*, we collect those items that are already rated by the user *u* (line 4). *selectedItems* is the set of items that we get by the intersection of *clusterTopN* and *I*. This is the line 5 in the algorithm. Basically, we want only those items rated by the user *u* which are similar in context to the item in question. This set, instead of the complete set of items rated by the user *u*, is then used to calculate similarities (lines 6-10). Also, we only want those items that are positively correlated to item *i*. Hence, those items that have a negative similarity value are not selected. The algorithm to calculate similarity between two given items is shown in figure 2.5. At this point we pick the top *k* items most similar to item *i* to put them in a set denoted by $S^k(i,u)$,read as, a set of *k* items rated by the user *u* most similar to item *i* (line 11). In lines 12 through 16, for each of the items *j* in this set, a baseline prediction is calculated denoted by b'_{ui}. The baseline predictions calculated here are adjusted by using the set *selectedItems* as explained earlier in algorithm 1. This calculation is detailed in the algorithm 4 as shown in figure 3.4. The variable *numerator* in the line 14 is the weighted sum of actual ratings by the user to the item *j* subtracting the adjusted baseline predictions. The sum of all similarity values is represented by *denominator*, line 15. The predicted rating, $r'_{ui,}$ is then estimated as the adjusted baseline prediction, b'_{ui}, plus the weighted average of the deviation of actual rating from the adjusted baseline predictions.

Let us consider the time complexity of this algorithm. We can infer from the algorithm given in figure 3.8 that the complexity of *clusterData* is $O(nK^2)$, where *n* is the number of items in the dataset and *K* is the number of centroids. Next, each of the algorithms *whichCluster* and *getTopN* perform a linear search and hence has a constant time complexity. Now, the algorithm

similarity is executed as many times as the size of the set *selectedItems.* As we saw in figure 2.5 in the previous chapter, the time complexity of *similarity* is O(N) where N is the number of users in the dataset. This is executed as many times as there are elements in the array *selectedItems.* This set will always have elements much fewer than the total number of items in the dataset. Hence the complexty so far for this process is O(N). The algorithm *Baseline* is O(n) as we can see in figure 3.4. Hence we can say that the overall computational complexity of this algorithm is $O(nK^2)$.

Algorithm 4: Baseline(u,i)

Input: user *u*, item *i*.

Output: baseline prediction b_{ui}

1: $\mu \leftarrow$ overall average rating.

2: array U(i) $\leftarrow$ users who rated item *i*.

3: **for each** p ϵ U(i) **do**

4: numerator1 += r_{pi} - μ;

5: **end for**

6: η = |U(i)|;

7: b_i = numerator1 / (λ_2 + η);

8: array I(u) $\leftarrow$ items rated by user *u*.

9: array TopNI(u) $\leftarrow$ top N items from the cluster to which item *i* belongs.

10: array selectedI(u) $\leftarrow$ I(u) $\cap$ TopNI(u)

11: **for each** q ϵ selectedI(u) **do**

12: numerator2 += r_{uq} - μ - b_i;

13: end for

14: η' = |selectedI(u)|;

15: b_u = numerator2 / (λ_3 + η');

16: $b_{ui} = \mu + b_u + b_i$;

17: return (b_{ui});

Figure 3.4: Algorithm to calculate adjusted baseline predictions.

The time complexity of the algorithm shown in figure 3.4 is O(n) where n is the number of items in the array *selectedI*. The for loop in lines 11-13 runs only as many times as there are items in the array *selectedI*. Note that he size of this array will be much less than the total number of items in the dataset and can never be equal to that.

The table in figure 3.2 should be examined in conjunction with the table in figure 3.5**.**

Movie id	Title	Genre	Rating(1-5)
1	The Hills Have Eyes	Horror, Thriller	4
2	The Ring	Horror, Mystery, Thriller	4
3	The Texas Chainsaw Massacre	Horror, Thriller	5
4	The Grudge	Horror, Mystery	4
5	Paranormal Activity	Horror, Mystery	4
6	Indiana Jones and the Kingdom of the Crystal Skull	Action, Adventure	4
7	Tangled	Animation, Comedy, Family	3
8	Brokeback Mountain	Drama, Romance	4
9	Happy Feet	Animation, Comedy, Family	2
10	House of Wax	Horror, Thriller	3
11	Paranormal Activity 3	Horror	?

Figure 3.5: Table of movie ids and their details with ratings given to each by the user *u*.

Let's walk through this example to get a better understanding of the ClusCor algorithm. As depicted by the table in figure 3.2 and as is evident with the genre information in the table in figure 3.5, movies with ids from 1 to 5, all fall in the same category of movies. It is safe to say that the category A, as mentioned before, is a combination of "Horror, Mystery and Thriller". The rest of the movies, though not all similar to each other, can be said to fall in another category B. Category B can safely be thought of as "Not A". Now, in order to predict a rating for the movie id 11, i.e. Paranormal Activity 3, the Correlation algorithm seeks to calculate the predictions by including, not necessarily but possibly, all the 10 movies this user has rated. Intuitively, movies from 6 to 10 being from a different genre category are wrongly influencing the caliper when the movie to be predicted does not belong to the same category. That is essentially to say that for predicting a rating for Paranormal Activity 3, Happy Feet should not have a say.

3.3 Clustering the Dataset

As the first step to predict a rating, we cluster the dataset with the given number of clusters. The choice of number of clusters is a subjective call. There may never be a correct number. It depends and varies with the nature and size of data to be clustered. There are algorithms that judge a given clustering and present an idea of how good the clustering is. But mostly it is subjective and data dependent.

We seek to cluster the dataset we are working with, the Movielens dataset[6], based on its genre information. This is a nominal/categorical data representing what all genres a movie belongs to. It is nominal because each of the categories is independent. There is no ranking or comparison "lesser than" or "shorter than". Just like colors, it is meaningless to say, for example, that blue is "greater than" white.

Seen as a vector of 19 values, a typical vector pertaining to genre information in our dataset can be represented as shown below. The 19 corresponding genres are shown in figure 3.6.

0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

A ‘1’ indicates that the movie this vector represents belongs to that genre. If the cells here are numbered from 1 to 19 starting from the very left, the table below shows which cells represents what genre.

Cell number	Genre
1	Unknown
2	Action
3	Adventure
4	Animation
5	Children’s
6	Comedy
7	Crime
8	Documentary
9	Drama
10	Fantasy
11	Film-Noir
12	Horror
13	Musical
14	Mystery
15	Romance
16	Sci-fi
17	Thriller
18	War
19	Western

Figure 3.6: Genre information.

As explained earlier in this thesis, we seek to cluster the data in a fuzzy fashion such that each item belongs to a cluster with a certain degree of membership representing how strongly it belongs to that particular cluster. Note again that for each item, the DOM values sum up to 1.

There are other algorithms proposed like the Fuzzy C-means algorithm. None of these algorithms were a suitable fit. The reason central to this is that the 19 categorical values that represent the data form a category in combinations of one or more of the values. Considering each alone themselves was not a good premise. The centroids of the clusters were to be fuzzy

themselves. Kim et. al. [17] proposed an algorithm to cluster categorical data in a fuzzy fashion. They proposed to do so with centroids being fuzzy themselves. For example, for the genre data a typical centroid can be represented as shown below in figure 3.7.

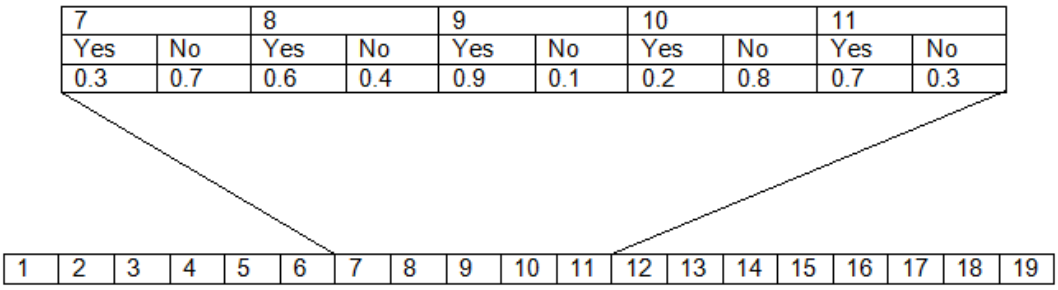


Figure 3.7: Representation of a typical genre centroid

The table above depicts that the centroid this table represents belongs to the genre field 7, i.e. Crime, with only a 0.3 parts out of 1 of the total items this centroid encapsulates in its cluster. Similarly, it suggests that the centroid is highly inclined towards the genre 9, i.e. Drama (0.9 parts).

The algorithm we employed for clustering is shown in figure 3.8.

Algorithm 5: ClusterData(dataset, numberOfClusters)

Input: dateset, numberOfCentroids

Output: centroids

1: centroids ← **chooseInitialCentroids**(dataset, numberOfClusters);

2: membershipTable ← **generateMembershipTable**(centroids, dataset);

3: while(terminationCondition == False) **do**

4: centroids ← **updateCentroids**(centroids, membershipTable);

5: **if**(terminationCondition == False)

6: membershipTable ← **generateMembershipTable**(centroids, dataset);

7: **end if**

8: end while

9: return (centroids)**;**

Figure 3.8: Fuzzy clustering using fuzzy centroids [17]

This algorithm works in following steps:

1) Given the number of clusters, choose initial centroids to begin with. These centroids are fuzzy as explained before.
2) Generate the membership table which holds the information pertaining to which item belongs to which cluster by what degree of membership. This is discussed in the next few pages of this thesis. Mathematically, degree of membership can be represented as,

$$\mu_{ip} = \left(\sum_{j=0}^{k} \left(\frac{\text{distance (i, p)}}{\text{distance (j, p)}} \right)^{1/(m-1)} \right)^{-1}$$

Where μ_{ip} is the degree of membership with which the item *p* belongs to the cluster *i*.

distance(i, p) gives the distance or measure of dissimilarity, between the point *p* and the centroid *i*. *m* here is the fuzziness factor, *k* the number of desired clusters.

3) Update the fuzzy cluster centroids after assigning membership values to each point. This way we align the centroids to be at the “center” of the cluster. We discuss this step in detail later.
4) If the centroids do not change significantly then stop. Otherwise, go to step 2.

Let us consider the computational complexity of this algorithm. As is evident from the description of the algorithm, its overall complexity depends on the complexity of the *generateMembershipTable* algorithm. We can see in figure 3.9 that its complexity comes out to

be O(nK^2) where *K* is the number of centroids and *n* is the number of items in the dataset. Complexity of the algorithm chooseInitialCentroids is O(*1*), i.e. constant. Algorithm updateCentroids is O(nK) complexity. Hence we have the overall complexity of clusterData to be O(nK^2).

3.4 Generate Membership Table

One of the steps in the algorithm explained above is to generate the membership table. We discuss the algorithm for the same now.

Algorithm 6: generateMembershipTable(centroids, dataset, m)

Input: centroids, dataset**,** fuzziness factor m.

Output: membership table.

1: for each *i* ϵ centroids **do**

2: $\mu_{ip} = 0$;

3: **for each** p ϵ dataset **do**

4: **for j=0 to j=**centroids.size **do**

5: μ_{ip} += [**distance**(i, p) / **distance**(j, p)]$^{1/m-1}$;

6: **end for**

7: $\mu_{ip} = 1/ \mu_{ip}$;

8: id = “i|p”;

9: value = μ_{ip};

10: membershipTable.add(id, value);

11: **end for**

12: end for

13: return (membershipTable);

Figure 3.9: Algorithm to generate the membership table.

Figure 3.9 shows the algorithm we use to generate the membership table. This table holds information as to with what degree of membership an item belongs to each cluster. Reader is reminded that the sum of these DOM values should be 1 across the clusters. Also, it should be clear that an item can have a very high DOM (as high as 0.9) with which it belongs to a particular cluster and yet have a non-zero DOM for other clusters.

The algorithm is based on the mathematical representation as depicted in the step 2 of the clusterData algorithm and as shown below.

$$\mu_{ip} = \left(\sum_{j=0}^{k} \left(\frac{\text{distance (i, p)}}{\text{distance (j, p)}} \right)^{1/(m-1)} \right)^{-1}$$

Let us consider the complexity of this algorithm. Let the number of centroids be *K,* the number of items be *n,* then the computational cost of calculating the distance of each item from the centroids will be O(1) as seen in line 5. This is inside the for loop in line 4 which runs *K* number of times hence making the complexity O(*K*). This is inside another for loop (line 3) that runs as many times as there are items in the dataset, i.e. *n*. Hence the complexity till this step becomes O(*nK*). Now, this for loop is inside the outer for loop beginning in the line 1 and runs only K number of times making the total computational complexity of this algorithm as O(nK^2).

3.5 Update Centroids

Updating centroids is an important step. The idea is to pick a new centroid for each cluster which better represents the items it encapsulates, quite like the k-means clustering algorithm

[18]. Difference here is that the centroids we have are fuzzy centroids. Suppose a typical centroid is represented as:

$V = \{v_1, v_2, \ldots v_l, \ldots v_p\}$,

and a typical item as:

$X = \{x_1, x_2, \ldots, x_l, \ldots x_s\}$.

Each of the fields (attributes) that constitute a point in the dataset can take one of the several different values. For example, x_l may take one of the t different values from the set $\{a_1, a_2, \ldots, a_t\}$

In our case it is two different values an attribute can take -either 'yes' or 'no'.

Any attribute in the fuzzy centroid will have a fuzzy value or weight for each of the many different values that the attribute can take. For example,

$v_l = w_1, w_2, \ldots w_t$ where w_1 represents the weight of the value a_1 so on and so forth.

Note that these weights add up to 1.

v_l is updated by determining $w_l^{(t)}$ for $1 \leq t \leq n_l$ as shown below[17].

$$w_\ell^{(t)} = \frac{\sum_{j=1}^{n} \gamma(X_{j,\ell})}{\sum \mu_{ij}^{m}}$$

Where,

$$\gamma(X_{j,\ell}) = \begin{cases} \mu_{ij}^{m}, & a_\ell^{(t)} = x_{j,\ell} \\ 0, & a_\ell^{(t)} \neq x_{j,\ell} \end{cases}$$

3.6 Get Top N Items

This method returns the top *N* items in a given cluster. Here the number *N* can be either of the following:

1) Actual number of elements provided at the beginning. This is not a very good heuristic as some of the clusters may have more number of items highly associated to it and not so much in some others.
2) A percentage of total items that form the cluster. The total items here can be all the items that have a greater DOM values than a given threshold towards that cluster.
3) All the items that have DOM values greater than a threshold. This way we need not limit the number of items included and yet get only those that actually represent the cluster.

For this research we have used the third criterion to pick the top N items from a cluster.

Chapter 4

Experiments and Results

In this chapter we discuss the results we obtained on our ClusCor algorithm, shown as algorithm 3 in figure 3.3. We compare the predictions thus generated with the ones we get from the Correlation algorithm, algorithm 1 in figure 2.3. We also compare it with the predictions generated if we use clustering alone for calculating similarities.

As explained earlier in chapter 3, the ClusCor algorithm first clusters the dataset based on contextual information about the items. These clusters are overlapping fuzzy clusters since one item (movie) can belong to multiple clusters at the same time. Each cluster center is also a set of fuzzy genre. If a rating *r* is to be predicted for an item *i* by a user *u*, the algorithm identifies the cluster to which the item *i* belongs with the highest degree of membership. A certain number of items from this cluster are then stored in a set. In another set we have all the items that the user *u* has already rated. We take the intersection of these two sets to get a set of rated items which are in the same cluster as the item to be rated. At this step we have gotten rid of those rated items whose vote, in our opinion, should not count in predicting the rating for item *i*. Now, using correlation to determine similarity, we pick the top 20 items from this set which are most similar

to the one to be rated (Correlation is determined between a movie from the set we have at hand and the one to be rated, for each movie in that set). Taking a weighted average of the ratings for these items, similarity being weights, we arrive at the predicted rating for item *i*.

4.1 Absolute Average Difference

To test the accuracy of predictions generated by each of the three algorithms, we calculated predictions for already rated items in the dataset modifying the dataset to have that item as not rated. We then take the absolute value of the difference in the actual and the predicted ratings one by one for each of the items the user has rated. The average of these values represents average absolute difference (AAD), in the predictions for that user. In this fashion we calculated 100 such values for randomly chosen 100 users. In figure 4.1, we show a snapshot of the table that holds these values for comparison.

	A	B	C	D	E	F
1	**UserID**	**# of Movies**	**Entropy of Ratings**	**Avg. Abs. Diff.(Correlation)**	**AAD (ClusCor 6 Clusters)**	**AAD (Clustering 6 Clusters)**
2	573	51	0.8325881	0.400537138	0.422201614	0.6922916
3	591	84	1.1607188	0.4166876	0.469988991	0.62142843
4	794	39	1.0100448	0.441488417	0.558212176	0.60482645
5	70	131	1.1983014	0.448844235	0.467819195	0.67819375
6	767	37	0.79716206	0.462766103	0.411904497	0.45495486
7	84	68	1.1657712	0.469471628	0.558678242	0.6826271
8	74	39	1.0498875	0.476696633	0.571131232	0.6100123
9	786	117	1.09935	0.479868323	0.542980384	0.6850746
10	94	400	1.3850791	0.483015998	0.533201946	0.7506834
11	85	288	1.201552	0.487861623	0.559607244	0.70226264
12	92	388	1.375124	0.498788961	0.568521931	0.7395584
13	793	55	1.1369952	0.505660049	0.604675867	0.7651517
14	764	109	1.2287332	0.514718254	0.584354791	0.7335658
15	389	271	1.3850131	0.515172929	0.588725756	0.8707461
16	784	39	0.98396146	0.51596586	0.517930817	0.81611717
17	399	319	1.1555653	0.517681454	0.559159249	0.63567865
18	788	249	1.2907536	0.526246751	0.562533759	0.69829774
19	393	448	1.1894864	0.526951842	0.568999979	0.67985904
20	90	300	1.1775986	0.529061558	0.569957199	0.7763492
21	380	162	1.2966661	0.532063339	0.600450959	0.72870487
22	766	175	1.2151608	0.537071559	0.618679085	0.6655598
23	87	211	1.2959874	0.539794461	0.616308266	0.7898106
24	576	36	1.2363237	0.543229462	0.60107281	0.7537036
25	398	172	1.3065196	0.54329498	0.667725971	0.87260616
26	577	188	1.2083291	0.546419855	0.595716419	0.67958
27	592	360	1.3895322	0.552224057	0.61352142	1.0717893
28	782	232	1.2812427	0.553747577	0.620435447	0.6837028
29	792	47	1.2196933	0.559938068	0.657209999	0.81800926
30	198	181	1.3512698	0.560663434	0.578691224	0.7787661

Figure 4.1: Average Absolute Differences in predictions over all rated items by different users.

To get an insight into what information the table similar to one in figure 4.1 has, we have plotted the points on a graph as shown in the figures that follow.

4.1.1 Experimental Setup

For the purpose of our thesis we have used the *MovieLens* dataset publicly available on the *GroupLens* website [1]. The datasets have changed recently and might give different results than what we have shown. The dataset comprises of 100K ratings given to 1681 movies by 943 users. The dataset is filtered to include only those users who have rated at least 20 movies. It is a randomly ordered, tab separated list of userid, itemid, rating and timestamp. We also have a dataset with contextual information about the items (movies); this is a tab separated list of movie id, movie title, release date, video release date, IMDb URL, unknown, Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western. The last 19 fields are the genres; a 1 indicates the movie is of that genre, a 0 indicates it is not. Movies can be in several genres at once.

We experimented on the number of clusters generated to see how it impacts the accuracy of predictions. For our algorithm the input parameters that vary are- number of clusters (or centroids), n and fuzziness factor m. It was observed that for the dataset we used, the clustering results were best when the fuzziness factor *m* was taken to be 1.5 and the number of clusters *n* as 6. Higher value of *m* increases the fuzziness and hence we get vague clusters.

4.1.2 Results

Figures 4.2 to 4.4 show the graphs we obtained with 5 clusters of items. On each of these graphs a point closer to x-axis indicates better performance.

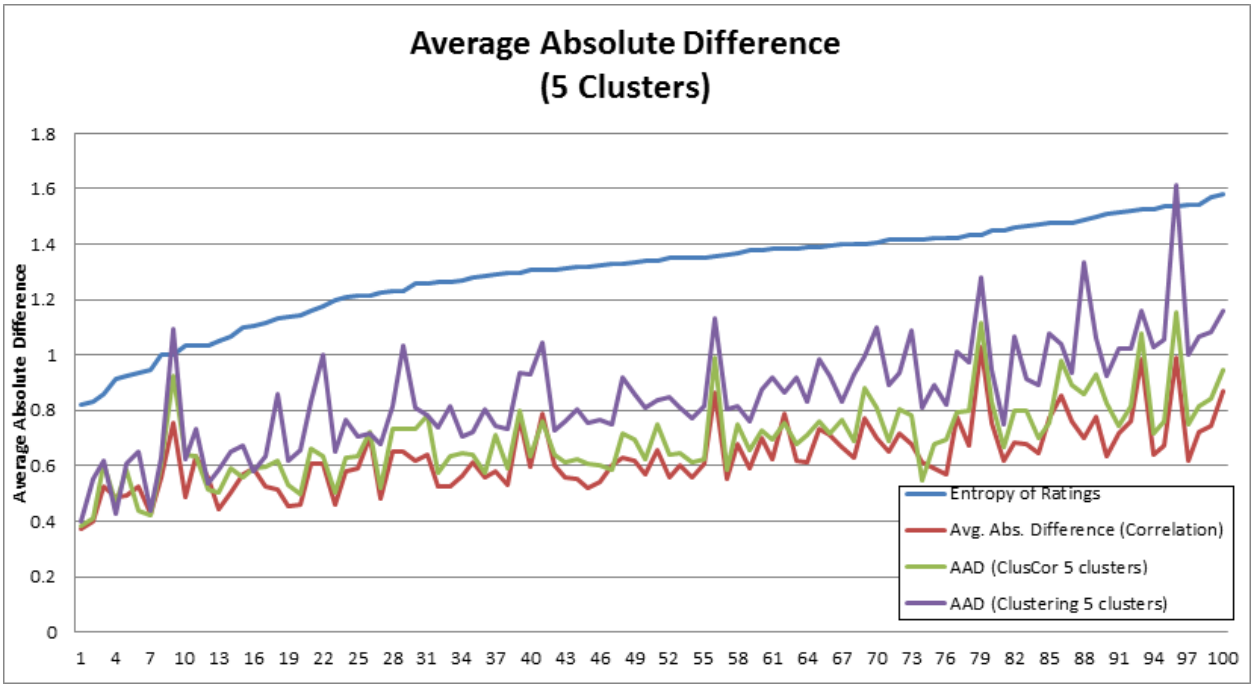


Figure 4.2: Average Absolute Difference values with 5 clusters and data sorted on entropy values.

The graph in figure 4.2 shows average absolute difference. Entropy is also plotted on the same graph just to work out a comparison. The entropy value is basically depicting the randomness of ratings given by a user. Hence a low value suggests that there is a consistency in the ratings that the user gives to different items. He/she has rated most of the items either average or above average or below that. A high value of entropy, however, indicates that the ratings are evenly distributed on the scale.

The graph in figure 4.2 suggests that as the entropy increases, the average absolute difference values for all the three algorithms also increase. This essentially means that all the algorithms perform worse for users having high entropy values of their ratings. The graph also indicates that predictions are worse in the case where similarities are calculated based only on clustering and most accurate, for most part, in case of using only correlation. It falls in between, except for a few places, for our algorithm which is correlation on top of clustering. It must be noted that ClusCor algorithm does much better than just clustering and in a few cases it actually performs

better than correlation algorithm. A low AAD value is desirable as it shows that the algorithm predicted a rating closer to the one actually given by the user.

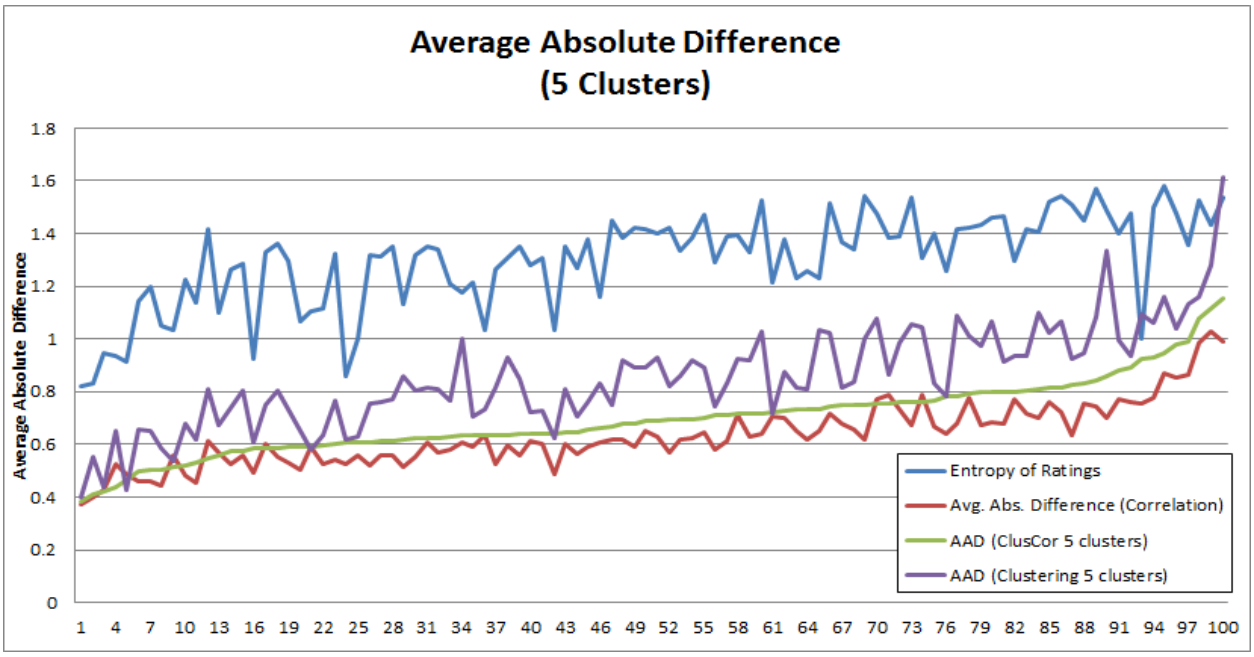


Figure 4.3: Average Absolute Difference values with 5 clusters and data sorted on ClusCor.

The graph in figure 4.3 shows average absolute differences sorted on values generated by the ClusCor algorithm. This makes it easier to notice those few cases where Correlation algorithm performs worse than the ClusCor algorithm. These are the ones where the red line crosses to go above the green line, away from the x-axis.

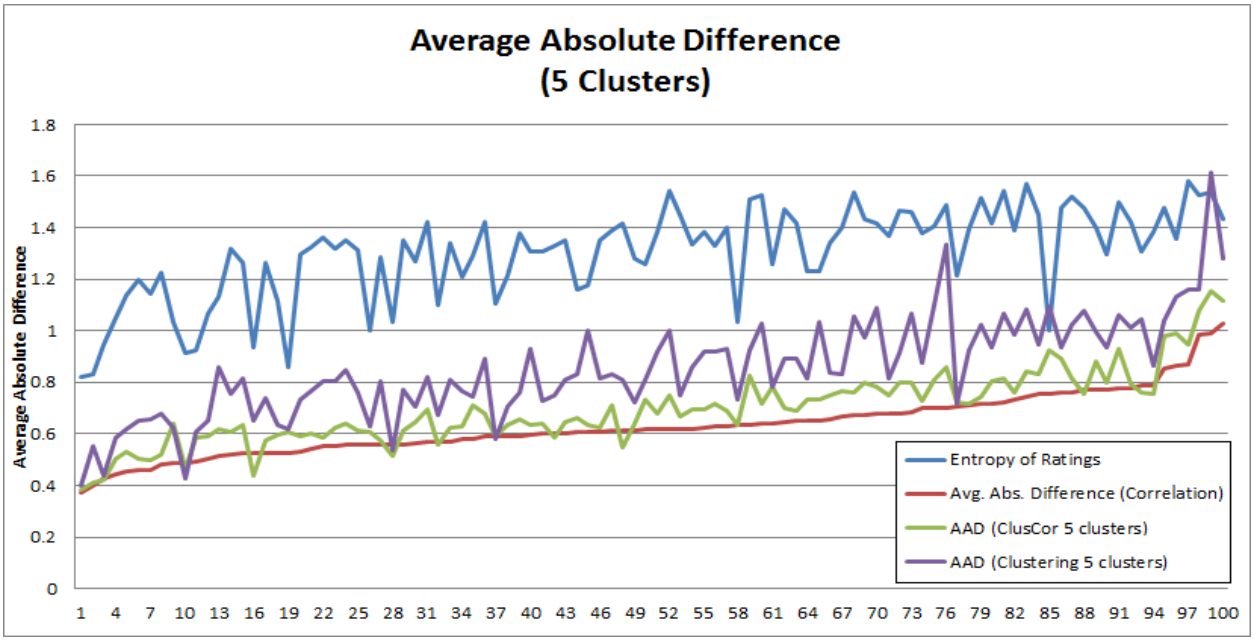


Figure 4.4: Average Absolute Difference values with 5 clusters and data sorted on Correlation.

The graph in figure 4.4 has average absolute difference values sorted on those obtained from Correlation algorithm. This simply makes it easier to follow that our algorithm does better only in a few cases which can be easily identified in this graph. The points where the green line (results from our algorithm) intersects the red line (results obtained from correlation algorithm) are the cases where our algorithm performs better than the correlation algorithm. Similarly, we plotted graphs for six and seven clusters as listed in the appendices. It is evident from those plots that the performance as a whole is not affected by varying the number of clusters. For each of these it takes about thirty minutes for ClusCor to produce results and Correlation takes about twenty minutes to complete. Clustering completes in less than three minutes.

Correlation B/w	5 Clusters	6 Clusters	7 Clusters
Entropy & ClusCor	**0.6806**	**0.6631**	**0.6967**
Entropy & Correlation	**0.6594**	**0.5908**	**0.6594**
Entropy & Clustering	**0.7537**	**0.7287**	**0.7327**
Correlation & ClusCor	**0.9360**	**0.9103**	**0.9387**
Correlation & Clustering	**0.8437**	**0.7096**	**0.7918**

Figure 4.5: Correlation values.

The table in figure 4.5 shows that there is a positive correlation between entropy and all the three algorithms. That is to say that the average absolute differences (AAD) of the predictions obtained from each of the three algorithms being compared, follow the trend exhibited by entropy. We can see that this correlation does not show any relation however to the number of clusters the items are divided in. Next, we can notice that AAD values obtained from Correlation algorithm have a significantly stronger correlation with those obtained from ClusCor algorithm than Clustering algorithm.

4.1.3 Deviation from Actual Ratings

The predictions that we generated for each item rated by a user were compared to the actual ratings that the user had provided. A deviation from these actual ratings tells how accurate the predictions have been.

In the figure 4.6, we show a comparison between the deviation observed in the results obtained from Correlation algorithm and our algorithm, the ClusCor algorithm.

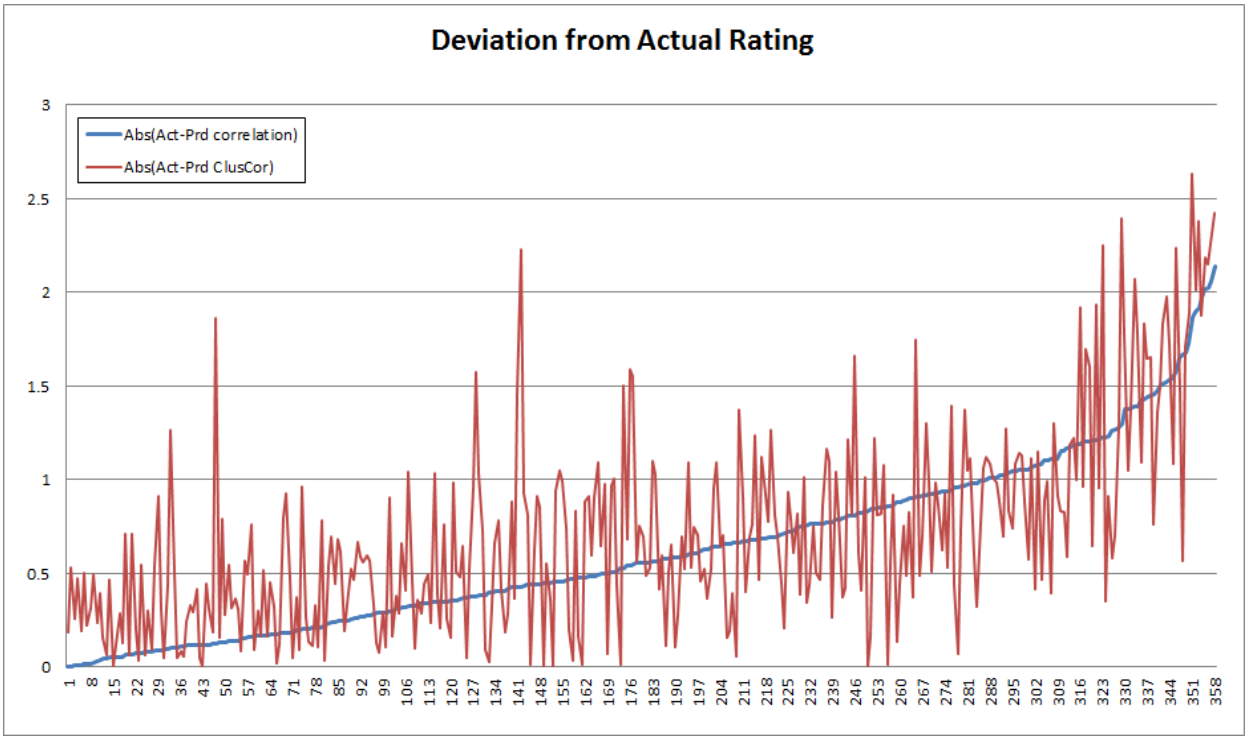


Figure 4.6: Deviation from Actual ratings for user 796 with 358 ratings.

The line in red represents deviation observed on ClusCor algorithm, and the blue line represents that observed on Correlation algorithm. Wherever the red line goes below the blue line, ClusCor should be understood as performing better than Correlation. In this particular case ClusCor does better than Correlation in 141 out of 358 predictions. This makes 39.38% cases.

We saw in earlier graphs that Correlation, except for a few cases, came out to be a clear winner (Figure 4.6) over ClusCor. The graph in figure 4.6 shows that if we look at the predictions for each item by a user, ClusCor performs better in about 40% of the cases. This observation is for the user 796, chosen at random. Figure 4.7 shows a similar graph plotted for the user 798, again chosen at random. The red line represents the performance of ClusCor algorithm. ClusCor did better in 90 out of 235 predictions which is 38.29% of the cases.

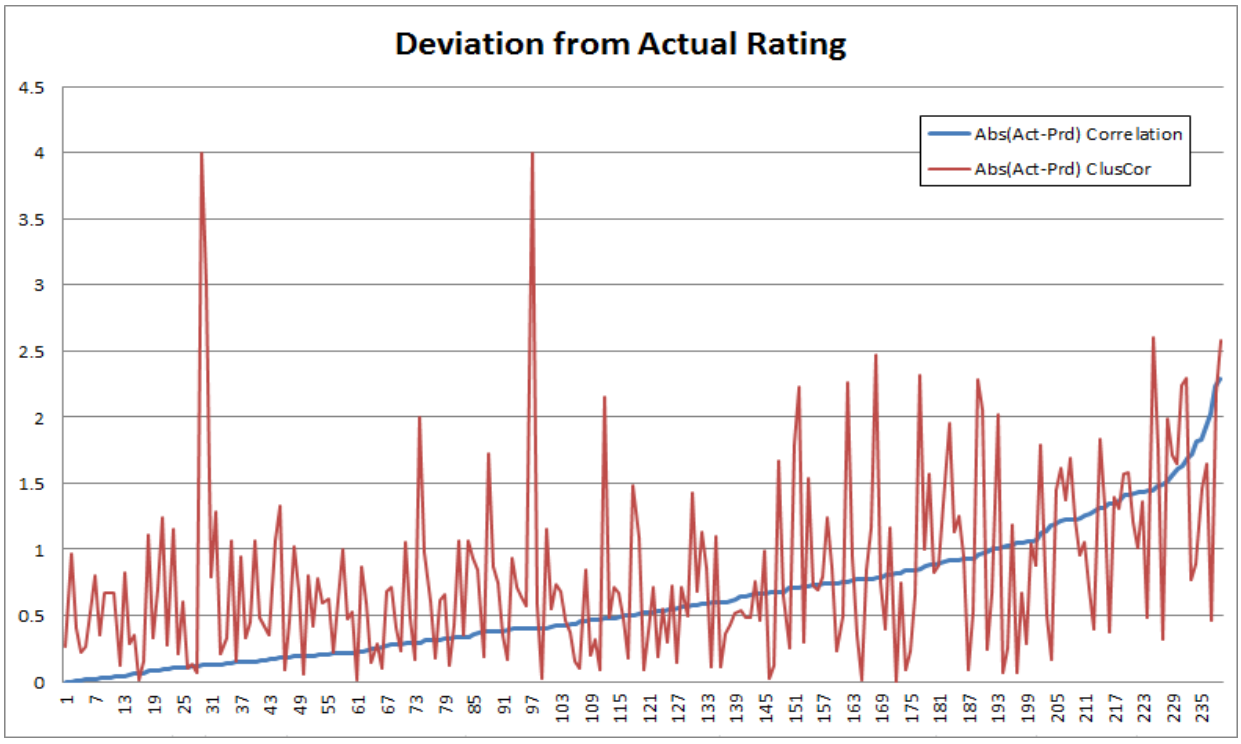


Figure 4.7: Deviation from Actual Rating for User 798 with 235 ratings

We also examined how these results relate with the movie entropies and figure 4.8 shows the graph. The results were sorted on entropy values and the graph suggests that the points where ClusCor performs better are randomly distributed throughout the curve.

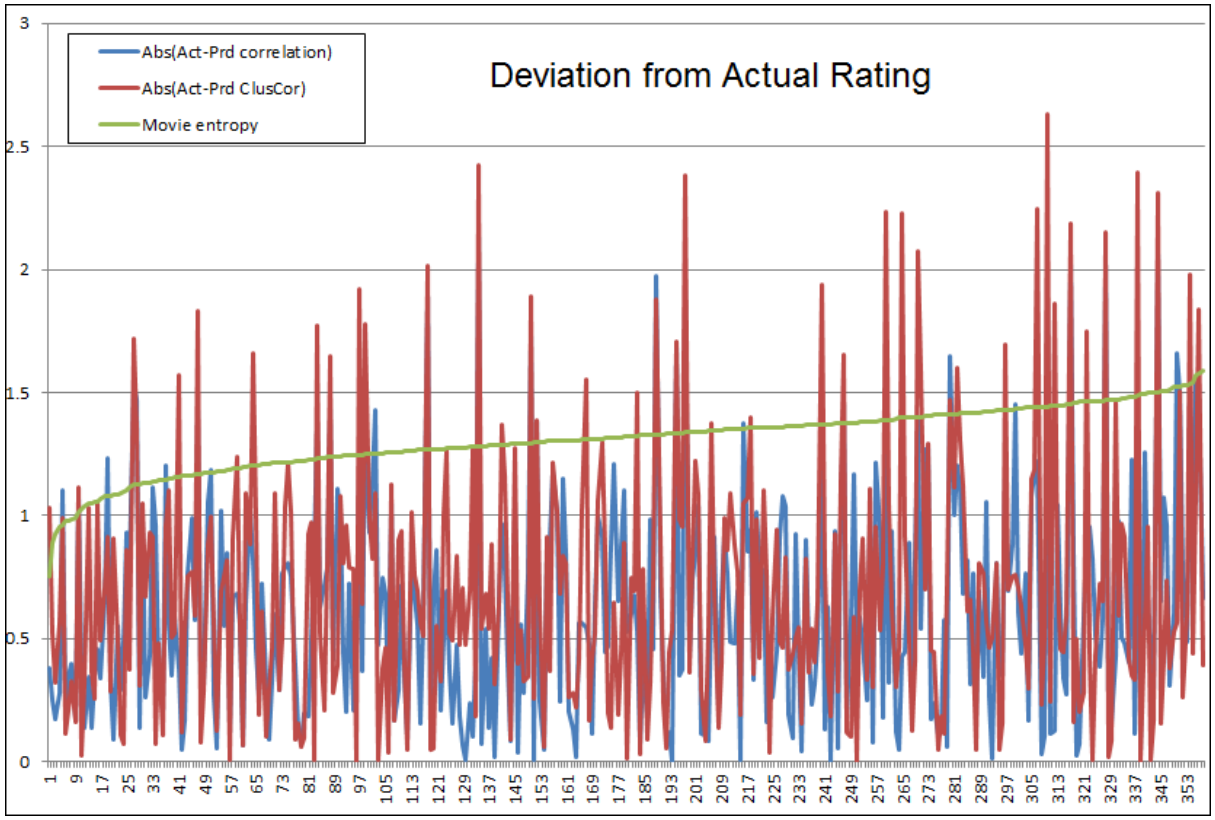


Figure 4.8: Deviation from actual rating relative to movie entropy.

4.2 The Entropy Test

Predictions are generated by taking the top K items most similar to the one for which a rating is being predicted. A weighted average on these ratings gives the predicted rating, similarities being the weights. In order to see if clustering would have an impact on the predictions, it would be of interest to find out whether the items that contribute towards prediction fall largely under one cluster. From the set of rated items sorted on similarity, most similar first, we take say top 20 items and see how many of these fall in which cluster. We then calculate the entropy

associated with this distribution to get a better idea. A low value of entropy would suggest that clustering can prove to have an impact on the predictions.

4.2.1 Entropy Calculation

For the running example, we randomly picked a user. This user had rated a total of 151 items in the dataset. We pick one item at a time from this list and make a prediction using the rest of the items. Each item is calculated a similarity value with the item to be predicted in the current iteration. The similarity values are based on Pearson's correlation coefficient. Since the dataset mostly has zeros, we rather take the empirical correlation coefficient. Once all the similarity values are generated, we sort the items on similarity values, most similar first. With this, we have also clustered all items in the dataset such that each item belongs to a cluster with a certain degree of membership. These membership degrees add up to 1 over all the clusters. From the sorted list of similar items we pick, say, top 20 items and see which cluster each falls into. We calculate the entropy associated to see if there is a situation where most of the top 20 items belong majorly to one cluster. Figure 4.9 shows a graph of the entropy values we got on our dataset.

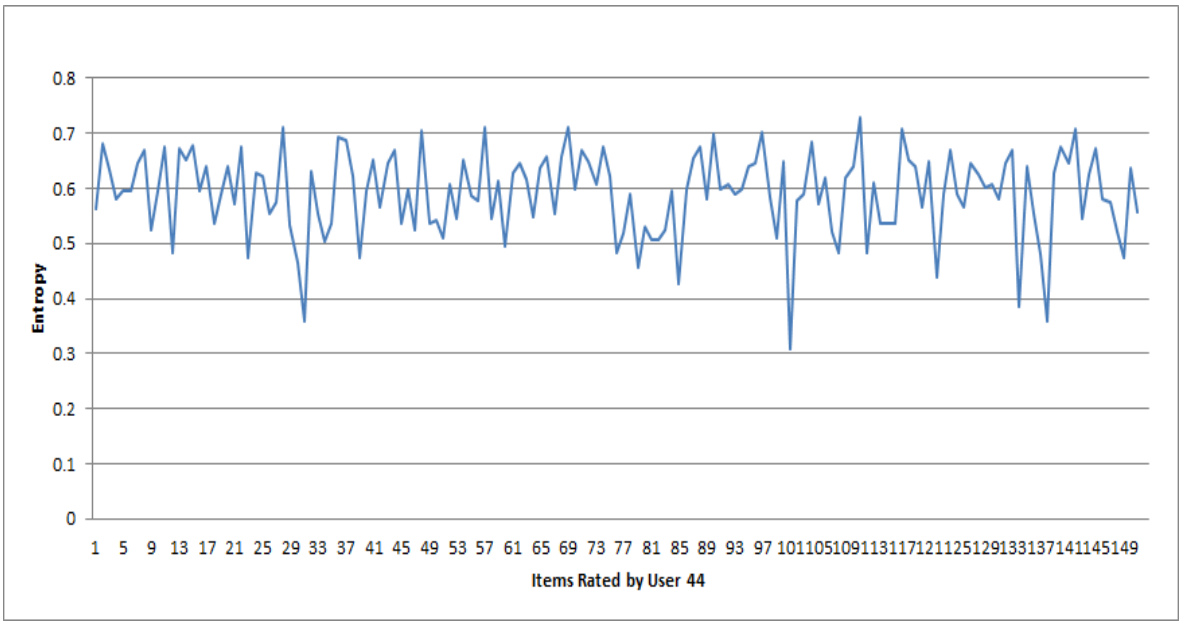


Figure 4.9: Entropy of distribution of top 20 most similar items into clusters. Ratings are taken from user 44 chosen at random.

Figure 4.9 reflects entropy values over 151 items (Xaxis) rated by user 44(chosen at random). The top 20 items most similar to the one being predicted were examined to see how many fell in which cluster. An entropy value was calculated for each such distribution which is what each point on the above plot depicts. The entropy was found to be high and hence it can be concluded that items mostly came from different clusters and choosing items from one cluster over the other to make predictions might not be beneficial.

Let's go through an example to better understand the calculations. We have 100,000 ratings on 1681 items by 943 users to work with. We clustered the items based on their genre information into 6 fuzzy clusters. It was observed that having 6 clusters and a fuzziness factor of 1.5 works the best. Consider, let's say, the nth item the user with id 44 has rated of the 151 rated by the user. Using correlation we pick 20 items most similar to this movie. Now, let's assume these 20 items fall in 6 different clusters as depicted by the figure 4.10.

Cluster #	# of Items	p_i
1	5	5/20
2	4	4/20
3	4	4/20
4	6	6/20
5	5	5/20
6	6	6/20

Cluster #	# of Items	p_i
1	1	1/20
2	0	0
3	17	17/20
4	1	1/20
5	1	1/20
6	0	0

Figure 4.10: Distribution of items in clusters as example to compare entropy values.

Entropy can be calculated using the mathematical relation

$$\mathbf{E} = (-)\ \Sigma\ p_i \ln(p_i)$$

p_i represents the probability which is number of items under this cluster divided by total number of items. Respective probabilities are shown in figure 4.10. So the entropy comes out to be

(-)[5/20 ln(5/20) + 4/20 ln(4/20) + 4/20 ln(4/20) + 6/20 ln(6/20) + 5/20 ln(5/20) + 6/20 ln(6/20)]

Entropy = 2.05.

The high value of entropy indicates that the items are spread out among the 6 clusters while a low value would have implied that most of the items fall largely under one of the 6 clusters.

Chapter 5

Conclusion and Future work

In this chapter we summarize the work done in the research. We also discuss the scope of improvements on the algorithm developed along with future works.

5.1 Conclusion

There have been many different algorithms proposed in the past to improve the predictions generated regarding the ratings given to items. As we learned in this thesis, the algorithms proposed so far would mostly form one of the three categories of recommendation systems, viz, Content-based filtering recommendation systems; Collaborative Filtering based recommendation systems or Knowledge-based recommendation systems. We also learned that of the three types of recommendation systems, except for certain specific contexts, the second kind mentioned above proves to show better results. A common algorithm that we set out to improve is based on Collaborative Filtering technique and is a neighborhood model based implementation. This algorithm, as we examined in detail in chapter 2, is built on Pearson's correlation coefficient. A user's potential rating for a movie will be determined based upon the ratings of all other movies by those users whose ratings have a strong correlation with the movie in question. It is seen to work well but there was obvious scope of improvement which became the motivation to develop the ClusCor algorithm as discussed in chapter 3.

Our hypothesis is that we should restrict the computation of the potential rating to only those movies that are in some way similar to the movie to be rated. We form clusters of movies according to the genres to which they belong. As is obvious, a movie can be a combination of several different genres. Hence a movie in the dataset can be seen to belong to multiple clusters. We select only those movies for computing a potential rating that belong to the same cluster and also have a strong correlation with the movie to be rated.

We have used fuzzy clustering of the movies in the genre space to create overlapping clusters of movies. This is to capture the intuitive notion that each movie can be classified as belonging to different genres to differing extents. We have used empirical correlation, which is in turn based on Pearson's correlation, to determine the movies that are strongly correlated with the movie to be rated. We performed experiments as discussed in the chapter 4 and analyzed results. The results show that in some cases our algorithm performs better but in some other cases it is seen to perform worse than the purely correlation based algorithm, but it always performs much better than the purely clustering based algorithm. It can be summed up as:

a) As entropy for user ratings increases, it was observed that the performance by all the algorithms worsens.
b) Our ClusCor algorithm comes very close in performance to the one that uses purely correlation and performs better in some cases.
c) Our proposed algorithm always performed much better than the one that uses clustering alone.
d) Although it was observed that our algorithm didn't outshine the one that uses purely correlation, it performed better in approximately 40% cases taken one user at a time.

Our hypothesis was that if we want to make a prediction for a movie then knowing its genre will help in getting a better prediction because we will then consider only other movies of the same genre previously rated by the user. But this hypothesis, it turns out, is not true. Movies that have high correlation with the movie for which a rating is to be predicted are better predictors than the smaller subset of the same genre movies. So, it is not necessary that extra piece of information about an item to be predicted will certainly improve the rating predictions.

5.2 Future Work

We have seen that the ClusCor algorithm performs better in certain cases, close to half of the cases, but worse in others. There is seen to have no obvious pattern as to which half is giving better results and which half worse. Having said that, the question whether there is a pattern is still open for investigation.

An improvement on the ClusCor algorithm could be to include weights from cluster memberships while filtering items based on fuzzy clusters to estimate similarities.

Bibliography

[1] “Grouplens.” Internet: http://www.grouplens.org, [Apr. 15, 2012].

[2] Y. Koren, R Bell. “Advances in Collaborative Filtering” in *Recommender Systems Handbook*, 2011. F. Ricci, L. Rokach, B. Shapira, P. B. Kantor, Ed. US: Springer, 2011, pp.145-186.

[3] H. J. Ahn. “A new similarity measure for collaborative filtering to alleviate the new user cold start problem”. *Information Sciences: an International Journal*, vol.178, pp. 37-51, Jan. 2008.

[4] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions”. *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 734-749, Jun. 2005.

[5] “Movielens.” Internet: http://www.movielens.org, [Apr. 15, 2012].

[6] “Gropulens Research.” Internet: http://www.grouplens.org/node/12, [Apr. 15, 2012].

[7] “Tutorial: Recommender Systems.” Internet: http://www.recommenderbook.net/teaching-material/slides, [Apr, 2012]

[8] P. Cremonesi, A. Tripodi, R. Turrin. “Cross-domain recommender systems.” in *Proc. ICDMW* , 2011, pp. 496-503.

[9] “Yahoo Music.” Internet: http://music.yahoo.com/, [Jun. 2012].

[10] “LinkedIn.” Internet: http://www.linkedin.com/, [Mar. 2012].

[11] J. Davidson et. al. “The youtube video recommendation system.” in *Proc. RecSys*, 2010, pp. 293-296.

[12] “Pandora Radio.” Internet: http://www.pandora.com/, [May. 2012].

[13] “Yelp.” Internet: http://www.yelp.com/, [Mar. 2012].

[14] “Knowledge based Recommender Systems.” Internet: http://www.cs.odu.edu/~mukka/cs795sum10dm/Lecturenotes/Day6/burke-elis00.pdf, [Mar. 2012]

[15] “Selecting and Applying Recommendation Technology.” Internet: http://maya.cs.depaul.edu/~mobasher/papers/recommender-technology-recoll08.pdf, [Mar. 2012]

[16] G. Ganapathy and K. Arunesh. “Models for Recommendation Systems in Web Usage Mining Based on User Ratings.” in *Proc*. *WCE*, 2011, pp. 525-530.

[17] D.W. Kim, K.H. Lee, D. Lee. “Fuzzy clustering of categorical data using fuzzy centroids.” *Pattern Recognition Letters*, vol. 25, pp. 1263-1271, Aug. 2004.

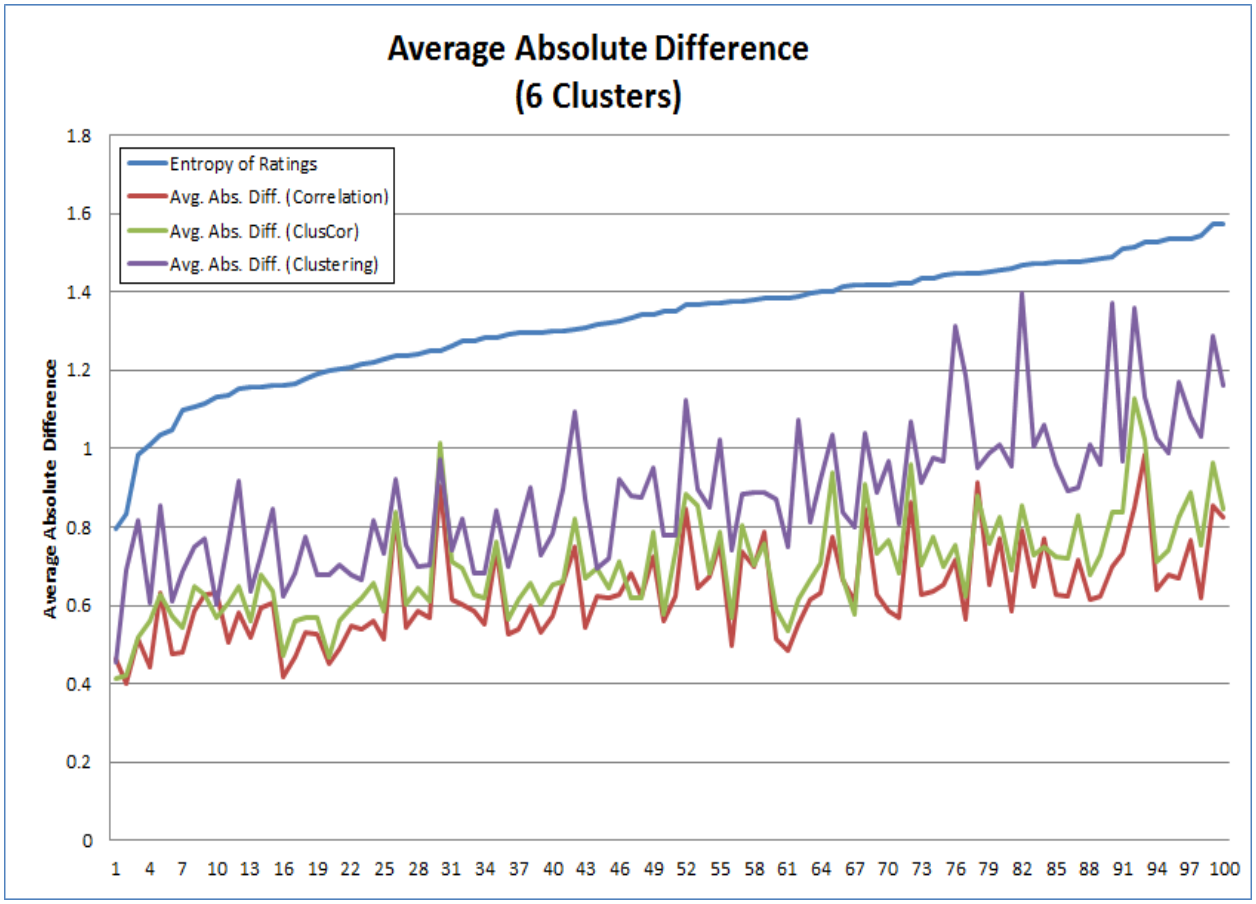
[18] “K- Means Clustering.” Internet: http://en.wikipedia.org/wiki/K-means_clustering, [May. 2012].

[19] G. Adomavicius et al. “Incorporating contextual information in recommender systems using a multidimensional approach.” *ACM Transaction on Information Systems,* vol. 23, pp. 103-145, Jan 2005.

[20] M. A. Tayebi et al. “CrimeWalker: a recommendation model for suspect investigation.” *in Proc.* RecSys, 2011, pp. 173-180.

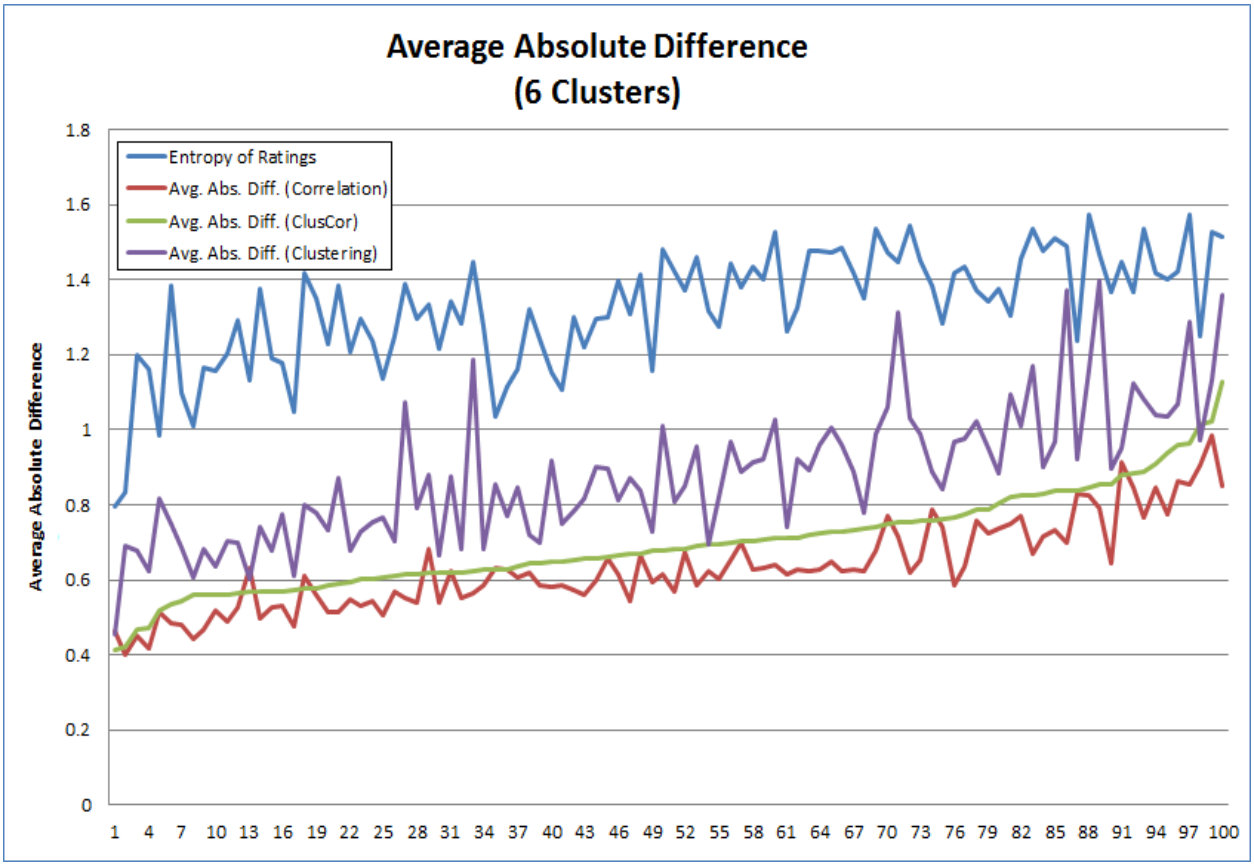
Appendix 1

Absolute Average Differences with 6 clusters and data sorted on Entropy values



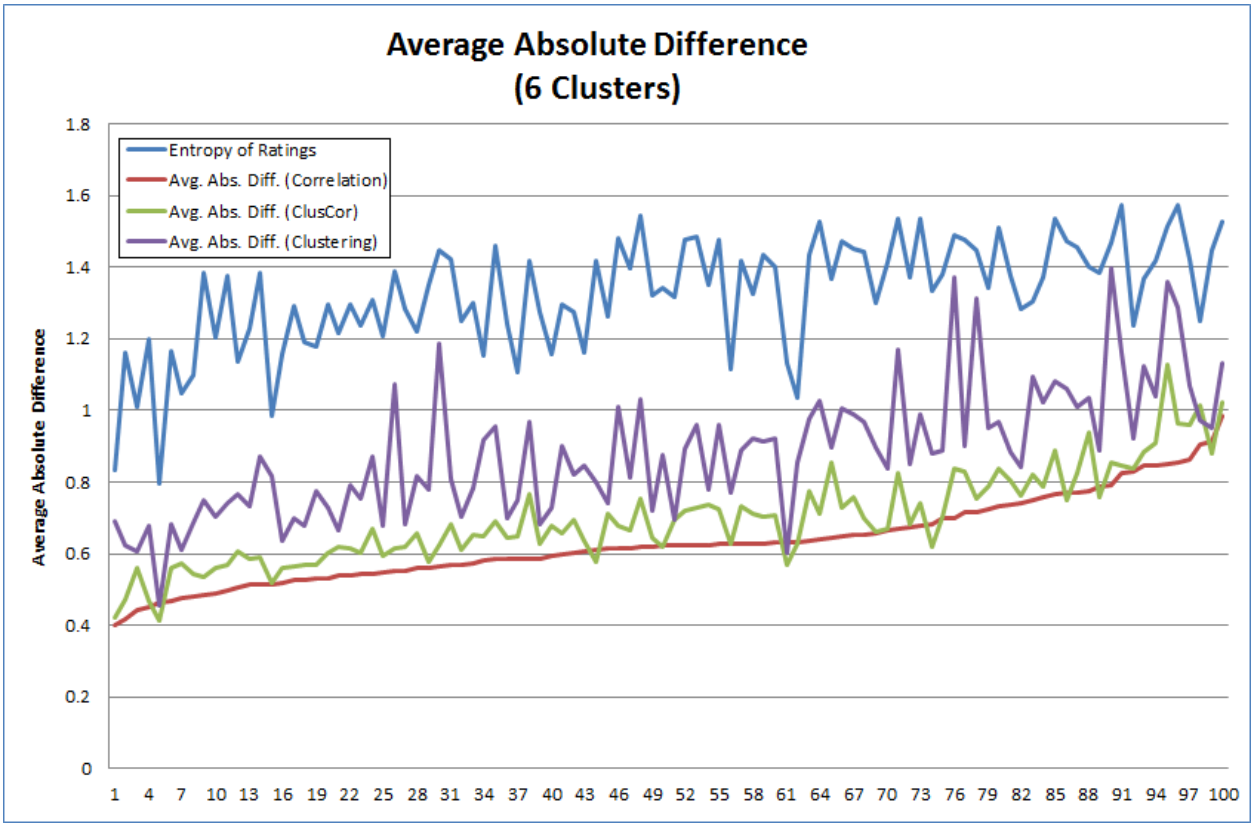
Appendix 2

Absolute Average Differences with 6 clusters and data sorted on ClusCor values



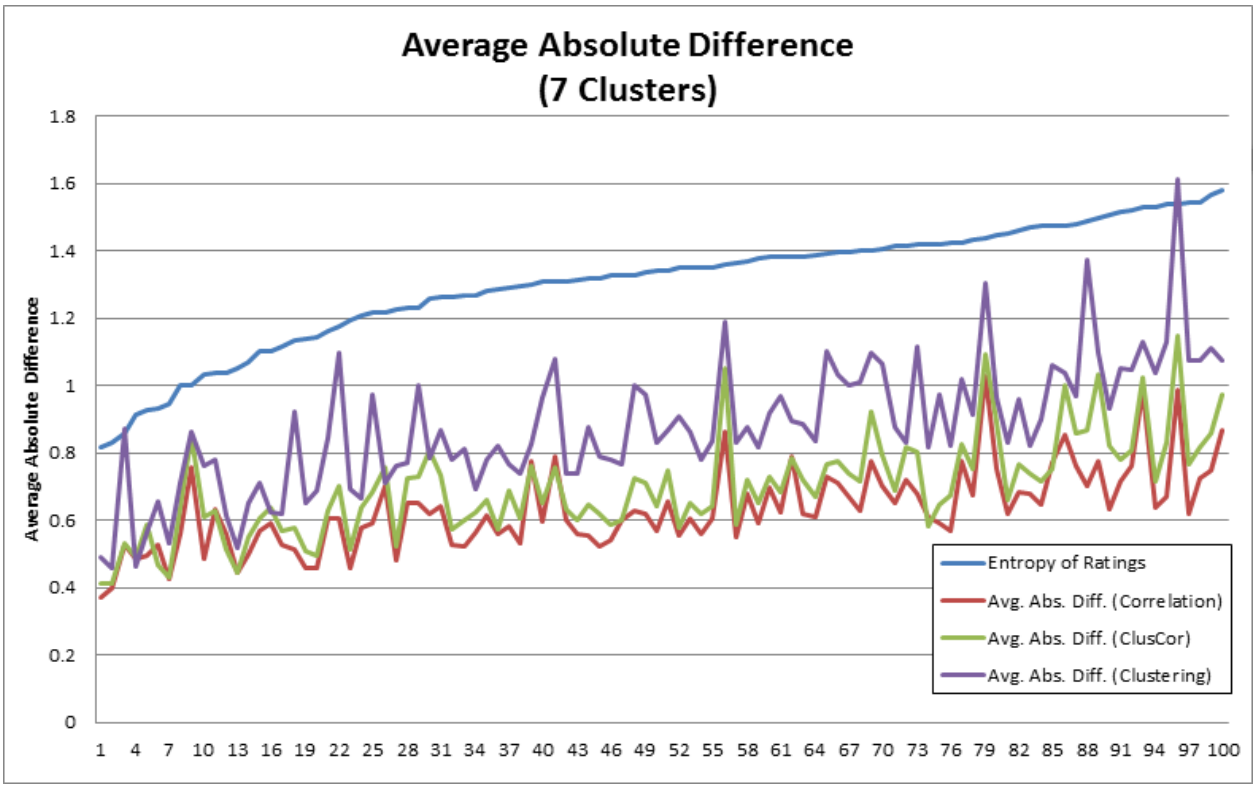
Appendix 3

Absolute Average Differences with 6 clusters and data sorted on Correlation values



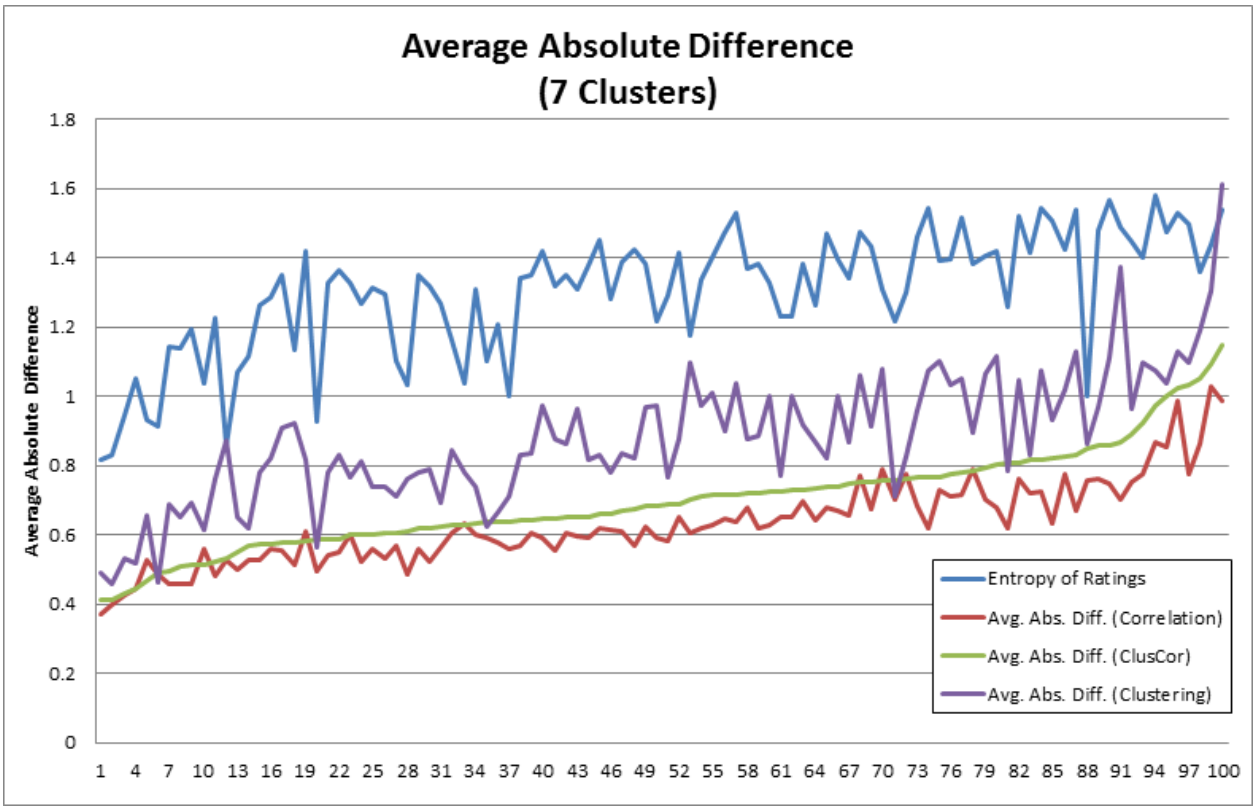
Appendix 4

Absolute Average Differences with 7 clusters and data sorted on Entropy values



Appendix 5

Absolute Average Differences with 7 clusters and data sorted on ClusCor values



Appendix 6

Absolute Average Differences with 7 clusters and data sorted on Correlation values