RecSys Project Report Template

James Dorrian

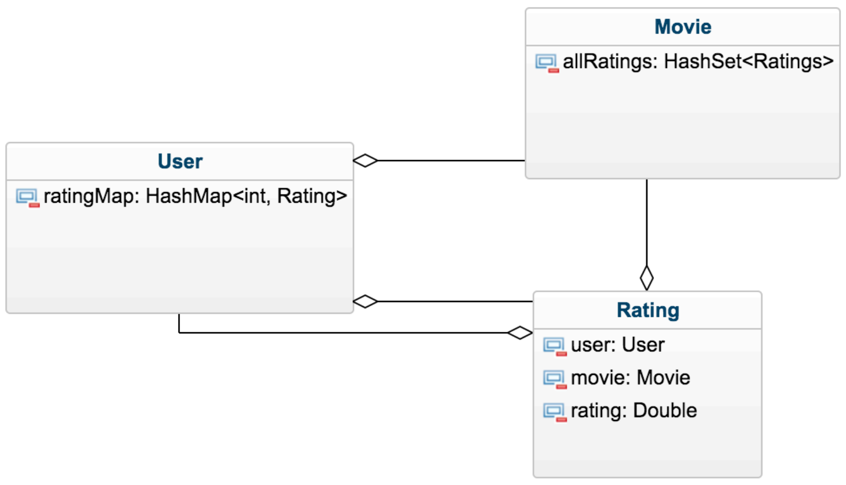
13369451
COMP30490

1 Introduction

Collaborative filtering is a method of making predictions whereby the taste of a user can be estimated by examining the tastes of other users who in the past have had similar tastes. The basic idea behind a recommender system is: by examining, selecting and aggregating large amounts of data from a sample dataset, relatively accurate predictions can be made on an individual's user preferences. The accuracy of these predictions can be improved dramatically using some well-designed collaborative filtering techniques. For the purposes of this project I used a 100k movielens database.

I made 3 basic classes in an effort to organize my code as clearly and succinctly as possible. To call the MovielensReader class I have a driver class, which also calls my part1, part2, part3 and part4 of the assignment. Each of these 4 parts in turn calls my L10 methods, which are located in my Evaluation file. Below is the basic relationship between my 3 object classes.

Fig. 1 - *UML outlining the basic interaction between my 3 most object classes*



A very basic collaborative filtering method called 'Vox Populi' meaning 'the opinions of the majority' was envisioned many years before the first computer was created. The basic concept here is that if people make guesses on a trivial question such as 'How many beans are there in this jar' they are equally likely to guess X beans too many as X too few. This applies to all of the values on either side of the actual number. If enough people guess the number of beans in a jar the wrong numbers somewhat cancel each other out and numbers closer to the mean converge. This gives us a number, which is often startlingly close to the actual number of beans in the jar. In the example cited in class, the average beat all the human guesses. This is a good example to showcase the power and accuracy of recommender systems. [1]

I implemented a very basic collaborative method for the purposes of my project called MeanAverageRating(). It was the most straightforward and simplistic of my collaborative filtering methods. It involved simply aggregating all the movie ratings for a given film (except the for user who's rating we are predicting) and dividing this number by the number of ratings for a given film (minus one). The results from this method were by far the least impressive of all our results with an average RMSE of 0.813386 and coverage of 99.859%. This isn't a terrible result but it leaves plenty of room for improvement. This method has an incredibly fast runtime of around 200 milliseconds on average.

The next collaborative filtering technique I employed is called MeanSquaredDifference. It is more complicated than MeanAverageRating() as it involves computing similarities between users, selecting a finite number of users to be added to the neighbourhood and then using only these neighbours to produce a prediction for a given user movie pair. This method can produce a RMSE value of 0.500369 and coverage of 84.20% where threshold = 40. (Threshold refers to the number of users in each neighborhood). Due to the number of loops and iterations of Movies, Users and Ratings the runtime is significantly slower taking around 1 minute to run on average (51283 milliseconds).

The final method I examined which is an extension and improvement on the Mean Squared Difference method is called Resnick's technique and implements Pearson's Correlation Coefficient formula to produce a more refined neighborhood and therefore more refined and accurate results. It is much more accurate than that of either of the two previous methods. With a threshold of 40 the RMSE was 0.315839 and the coverage was 51.51%.. Runtime was not too much slower and took an average of around 1 minute (57892 milliseconds).

It is my postulation that systems with neighborhoods work much better than those without and I will go on to show this in sections 3.4, 3.5 and beyond.

2 Collaborative Filtering

2.1 An Overview of Collaborative Filtering

Collaborative filtering is a useful tool for interacting with large and often complex data sets. These systems offer a method of personalizing data in a way that prioritizes items that may be of interest to the user. Personalized recommendations are much more effective and valuable to the user than non-personalized data as they can predict an individuals taste much more accurately than a generalized system for all users can. These systems are growing in popularity internationally and can be seen in use by massive corporations such as Netflix, Amazon, LinkedIn and Facebook.

Collaborative filtering works by taking large data sets and analyzing user preferences in order to predict as close an estimate as possible. One of the main benefits of a recommender system is its relative simplicity i.e. it does not rely on the computers ability to analyze complex data (machine learning). This means that it can accurately make predictions for a movie with no understanding whatsoever about what it is predicting. At a very basic level, the way it chooses what rating it will give is based on the idea that people who have had similar tastes in the past will have similar tastes in the future. This is a good hypothesis; particularly for our dataset as movie tastes are developed over a long period of time and are therefore not subject to rapid change.

Our system relies on the explicit ratings given by users i.e. a rating between 1 and 5. More complex recommender systems rely on implicit information gathering such as that seen on the Amazon web store when simply browsing products is enough for Amazon's algorithm to suggest similar products.

2.2 Approach 1 – Mean-Rating Prediction

In my code the Mean-Rating Prediction is contained in a class called 'MeanAverageRating'. It takes a movie and a user as input. The goal of this method is to take all of the ratings for this movie and use these values to predict the rating our user would give it.

To do this we create a list of all the ratings of that film. This list represents all of the ratings given for this film by all of the users in our sample set. It is important to note that this set of movie ratings includes the user who's rating we are trying to estimate.

To remove a user from this rating simply take the rating away from the total before division and then divide by the number of ratings – 1. As depicted in this code snippet below:

```
if (totalRatings.size() > 0){
    return ((avg-ratingRemove)/(totalRatings.size()-1));
}
```

2.3 Approach 2 – Distance-Based CF

The second approach I chose was the Mean Squared Difference method. It is a much more complex collaborative filtering method than Mean Average Prediction. It revolves around the notion of a distance metric. What this means is that it concerns itself with the difference between 2 user's ratings for a given film.

For this approach there is much more required than a single method. First we must consider the computeSimilarity() method. In this method we take 2 users and get all the movies they have in common. We then cycle through this list and get the rating the users have applied to each common film. We then take the distance between them (ensuring its always positive) and aggregate them. This number divided by the total number of common films gives us our similarity metric. This is the number that is used to decide whether or not the given user is going to be included in another user's neighborhood. If zero is returned from this function then that means that no common ratings between 2 users and therefore they cannot be in the same neighborhood.

I stored all of the distance values from computeSimilarity in an NxN matrix called ‘matrix’ and used a getMSD() method to retrieve the data and setMSD() method to call the above computeSimilarity and set the values for each user pair.

Having already processed and stored the similarity metric for each user pair choosing a neighborhood was very straightforward. It required us to choose a threshold (number of users in the neighborhood) and any users whose similarity metric was not inside the threshold range is excluded from neighborhood creation. It is important to add an additional check alongside the neighborhood check to make sure that there are enough ratings in the total neighbor set to add to the neighborhood. This check is shown below:

```
while (neighbourhood.size() < threshold && neighbourhood.size() < interim.keySet().size()){
    neighbourhood.add(interim.get(vals.get(0))); //the key to the interim hashmap is the similarity rating
    vals.remove(0);
}
```

Once we have our neighborhood we can easily make predictions for any user movie pair. From my Evaluation method (which is where my Leave one out loops are) I cycle through all of the users and all of the movies and get the predicted value for each movie a user has rated. I vary the similarity threshold of users to give me results across a 10-user neighborhood; 25-user neighborhood and 50-user neighborhood and these results are saved to individual CSV files.

Using these predicted values I can calculate the RMSE by taking the estimated value from the actual value and ensuring its positive use Math.abs (). The code snippet above shows us taking the minimum value available and adding it to the neighborhood list. This is because MSD measures the distance between 2 user’s rating and the smaller it is the more suited they are to be in each other’s neighborhood.

2.4 Approach 3 – Resnick’s Prediction Technique

The final collaborative filtering approach we examined was an approach called Resnick’s Prediction technique, which uses the Pearson Correlation coefficient to help calculate the similarity metric. It is different to the Mean Squared Difference technique in that it has more to consider than just the similarity between two users, it also takes the average movie rating of a user.

The main difference between the PCC and MSD is that PCC considers the user’s average movie rating. This can help avoid inaccuracies which arise from rating patterns in users, for example, it is important to be aware of a user who has a habit of always giving either a 4 or a 5 but who never rates films too harshly. Equally it is important to be aware of the opposite users who will always harshly rate titles constantly giving scores of 1 or 2 at most. In layman’s terms the difference between PCC and MSD compute similarity method is that the PCC considers how far the user’s average rating is from our target user’s average rating and weights the user’s rating accordingly.

Unlike MSD PCC works on a scal of -1 to +1 where -1 indicates a strong negative correlation to the user, i.e. their ratings in the past were as far apart as possible. Conversely, a PCC similarity metric of +1 indicates that 2 users could not be more similar in their ratings. In practice it is extremely rare that either a -1 or a +1 value be found between 2 users but the spectrum is very useful when evaluating whether or not 2 users are similar, and if they are, how similar. The below code snippet shows us selecting the neighborhood where the largest possible values are selected as the array is sorted in ascending order.

```
while (neighbourhood.size() < threshold && neighbourhood.size() < interim.keySet().size()){ //interim = list of all neighbours
    neighbourhood.add(interim.get(vals.get(vals.size() -1))); //the key to the interim hashmap is the similarity rating
    vals.remove(vals.size()-1);
}
```

For the purposes of this recommender system I used a threshold value that affects the size of the neighbor hood directly threshold=20 means that a neighborhood will have at least 20 neighbors (if there are adequate ratings available). Another option would be to use a similarity threshold which would create variable neighborhood sizes but which would involve me normalizing my MSD values between -1 and 1 to make fair comparisons.

Once PCC has been used to compute the similarity metric, we this this metric (stored in a matrix of dimension NxN where N=number of users) to compute our neighborhood. Unlike in MSD where we once used to check if our similarity ratings were less than a certain threshold we now check if our similarity is greater than the threshold (which must be less than 1) as the values are normalized.

3 Evaluation

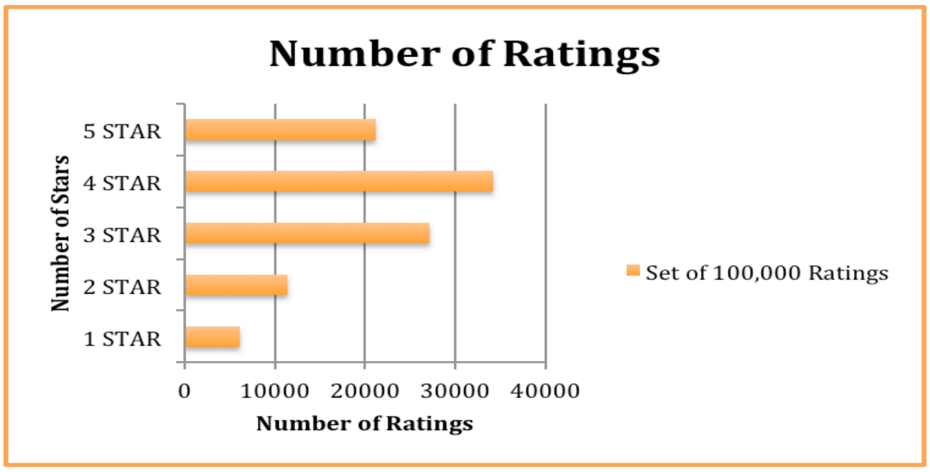
The evaluation is the most important and informative part of my report. I clearly showcase my findings and explain how and why I received these results.

3.1 Dataset

Our dataset contained 943 users, 1682 films and 100,000 ratings.

There were 6110 1-star ratings, 11370 2-star ratings, 27145 3-star ratings, 34174 4-star ratings and 21201 5-star ratings.

Fig. 2 – *Number of stars of each given rating in our movielens database*



Our data set density is *~6.3%* which is extremely sparse.

Data set sparcity is a huge problem for recommender systems and it has a huge impact on the accuracy of predictions. In the real world this problem is amplified by another problem associated with a lack of data called the ‘Cold start problem’. The term ‘cold start’ refers to the issue encountered when a new recommender system is put into use but due to the ‘newness’ of the system, it severely lacks the required data to populate the database and generate accurate predictions. [REF]

3.2 Methodology & Metrics

Leave one out (L1O) testing is the evaluation methodology we implemented in this project. It involves cycling through all of our users and their rated films and evaluating the values we would predict for a given user film pairing while not considering the pairs actual rating.
In Mean Average Rating we do this by aggregating *all* the ratings to find the mean. Our other techniques (MSD and Resnick) are *more complex* as we use neighborhood construction to determine which values to consider whether or not to consider a given user’s rating when calculating our prediction. [2]

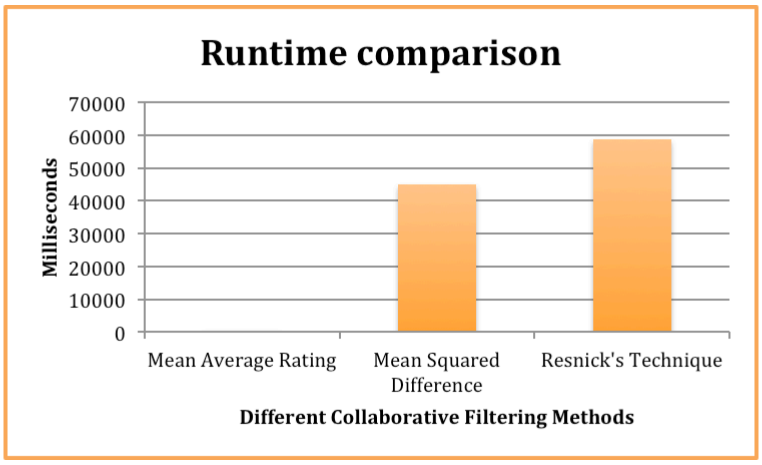
We used 3 main evaluation metrics in particular to evaluate our results:

1. Timing
2. RMSE
3. Coverage

3.2.1 - Timing

To determine the speed of our different programs it is necessary to implement timing functions, I did this in my Driver class by calling each of my L1O tests 10 times and getting the average. The timing of my three evaluation methods varies widely. First we have 246ms for Mean Average Rating, then 44972ms for Mean Squared Difference and finally 58734ms for Resnick’s technique. The main reason for such a large range of values is that Mean Average Rating is a simplistic collaborative filtering class containing only a single method, which is called statically. For Mean Squared Difference and Resnick I created fully instantiated classes. This technique and my method of storing similarities (NxN matrix) definitely reduced my codes efficiency. The reason I did this was so I could store and manipulate the data easily in a very well designed and logical manner. I was not too concerned with runtime because this was only an assignment, however, in the real world, runtime would be a massively important aspect of any recommender system.

***Fig. 3** - Runtime comparison of our 3 collaborative filtering techniques*

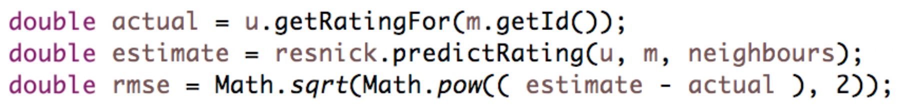


Please note that the data used for figures 3, 4 and 5 are all from the same run of the program where neighborhood size = 200.

Scalability is one of the biggest problems faced today by recommender systems. As the system grows, so too does the number of ratings, users and movies. When this happens, the neighborhood calculations become increasingly expensive in terms of processing power and therefore as a result, time. I outline steps that I would take to improve the runtime of my system if it were to be implemented in the real world in my concluding paragraphs. [REF TO SCALABILITY APPENDIX]

3.2.2 – RMSE – Root Mean Squared Error

The values obtained from each collaborative filtering method in regards RMSE helps us evaluate to what extent a recommender system can predict ratings of users. [3] Root mean squared error is calculated when we make predictions and compare them to our actual ratings. The actual ratings are omitted from the calculation as intended by the L1O technique. RMSE involves taking the estimated value from the actual value and squaring it to make sure its positive. Then it gets the square root of this number to bring the value back to normal. It is a good indicator of the accuracy of our results and the average of an L1O's rmse values can be used as an indicator of the predication accuracy of our collaborative filtering techniques. The formula can be seen at work in this code snippet: (where u = user and m = movie.)

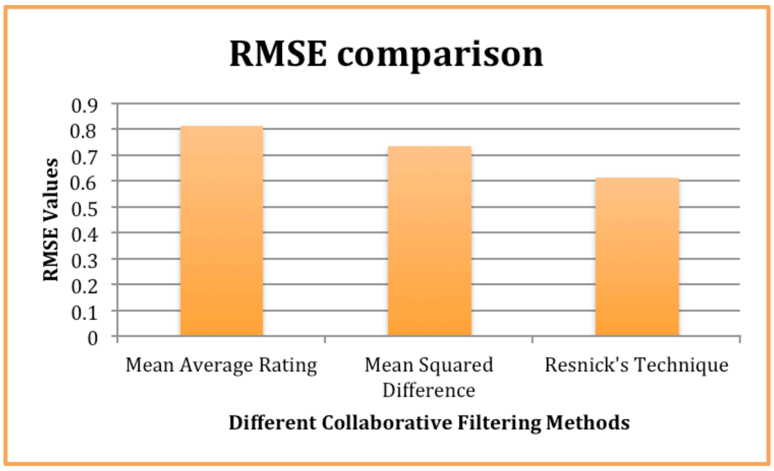


```
double actual = u.getRatingFor(m.getId());
double estimate = resnick.predictRating(u, m, neighbours);
double rmse = Math.sqrt(Math.pow(( estimate - actual ), 2));
```

This value is converted to a positive number because we are concerned with whether we guessed too high or too low but rather with the degree of incorrectness in our estimates.

The diagram below depicts the RMSE values obtained from Mean Average Rating, Mean Squared Difference and Resnick's technique where the neighborhood size for MSD and Resnick was set to 200. This high neighbourhood value allowed me to put the 3 figure on the same graph as coverage is almost identical. The exact coverage values are 99.859%, 99.238% and 97.317% for MAR, MSD and Resnick respectively.

Fig. 4 - *RMSE comparison of our 3 collaborative filtering techniques*



This figure proves one of our hypothesis' which states that collaborative filtering algorithms with neighbourhoods produce better results than those without neighbourhoods and furthermore, the more refined the creation of those neighborhoods the more accurate our predictions will become.

3.2.3 - COVERAGE

Coverage is the final metric we used to compare and contrast our data. Coverage refers to the number of titles which a prediction can be made for. I used a percentage to represent this in the overall evaluation of each collaborative filtering method. In general it is strongly correlated to the size of the neighborhood.

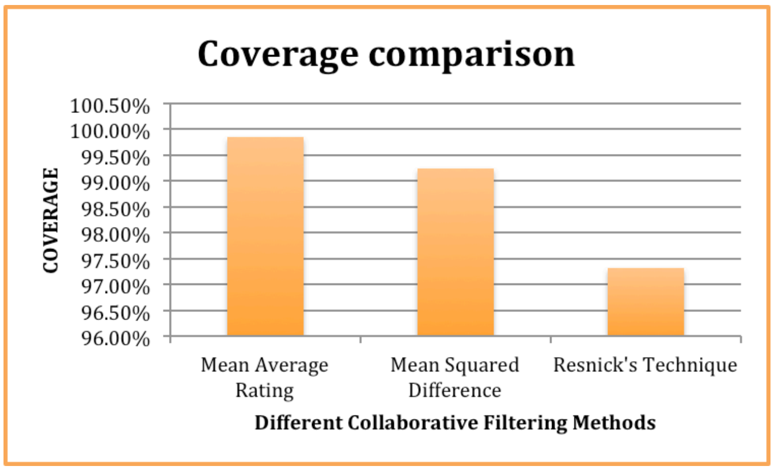
The logic behind there being a strong correlation between to the size of the neighborhood and the coverage is that a reduced number of members in a neighborhood means that there are less ratings from which to make an estimate of a users rating. However, it is important to mention that although smaller, more rigidly refined neighborhoods have much less coverage it is common for them to have a greatly reduced RMSE. This means that the prediction accuracy increases.

There are two generalizations can be made about the relationship between neighborhoods, coverage and RMSE:

1. Large unrefined neighborhoods have more coverage but worse average RMSE
2. Smaller more refined neighborhoods => less coverage but better average RMSE

Below is the coverage comparison between Mean Squared Difference and Mean Average Rating.

Fig. 5 - *Coverage comparison of our 3 collaborative filtering techniques*



3.3 Approach 1 Results – Mean Average Rating

Mean average rating was by far the least complex method of evaluating predictions for users implemented in this project. It did not involve computing similarity metrics nor did it involve computing neighbors or neighborhoods. It had the fastest runtime and very high coverage but did not have a particularly good RMSE values when compared to the other CF methods.

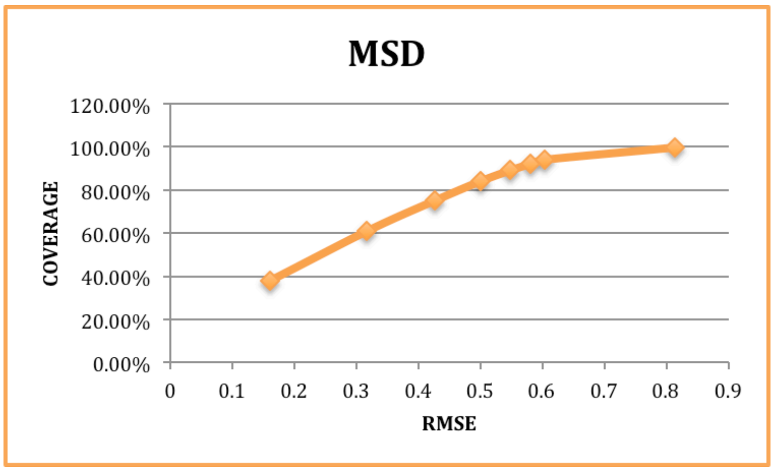
The coverage was 99.859% with the only titles in the entire dataset that could not produce a predicted rating were those with only 1 user rating. When it came to predict this user's estimated value we were unable to do so as nobody in the rest of the dataset had made a rating. A NaN (not-a-number) value was written to the csv file in this case.

The average RMSE value was 0.814535 meaning that on average our guess was 0.814535 from the actual rating. This isn't a terrible result but we must remember that this is just an average value. When I examined the results individually some of the estimates are very far from the mean. For example for user 405's rating on movie 851, this method's guess is 3.66666666667 from the actual value given by user 851. Large discrepancies like this are much more common in systems which do not employ a neighborhood system.
In summary, this is a good generalized approach and a brilliant starting point for building a recommender system. It is good for visualization of what we are trying to achieve and useful when examining areas for improving the system. However, I would not recommend using this system for making real world recommendations are there are too many outliers and the RMSE values aren't good enough.

3.4 Approach 2 Results

Mean Squared difference was the second approach I examined and the first, which involved neighborhood creation. It is a more personalized system than the Mean Average Rating and the results clearly show this. Below is a graph, which depicts the effect of neighborhood size on RMSE and coverage. For this I varied the neighborhood size from 10 to 80 in increments of 10. Each increment of 10 is represented by a point on the below scatter graph.

***Fig. 6** – Coverage and RMSE changes graphed for changing neighborhood size*



It is clear that the RMSE value increase as the size of the neighborhood increases. In section 3.2.3 I made a generalisation that when the size of the neighborhood is increased the coverage also increases. This is clearly displayed above with the ever increasing value on the y-axis. I also stated that as as the size of the neighborhood increased and we allow in members who are less and less similar to our target user, the RMSE increases. This is seen by the increasing values on the x-axis.

It is fair to say that coverage and RMSE are both strongly correlated to neighborhood size.

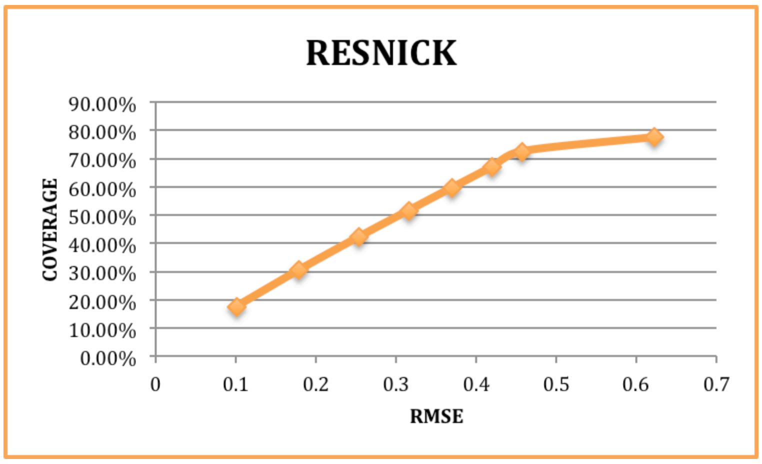
3.5 Approach 3 Results

As I touched upon before, Resnick's technique is more complex than the mean squared difference. Resnick considers the average overall movie rating of each user we consider when making a prediction and we can therefore scale a user's rating on a particular movie accordingly. For example, if a user always gives 4's and 5's for every movie we can scale down rating of 5 for a given movie X accordingly. This is why the RMSE is significantly lower in figure 7 than in figure 6.

Coverage is also reduced when compared to our MSD coverage values for neighborhoods of same size. This is because when we consider the use of the Pearson correlation coefficient's averaging technique employed when a generating similarity metric, it is more restrictive than MSD. For example, where a user has only ever rated 10 titles and they all have been 5 star ratings then this user's similarity metric will be much less than the value MSD would assign him. This statement assumes that the target user has rated a few of the 10 titles in question quite highly.

The takeaway point here is that once again we see that a more restrictive neighborhood computation has given us better coverage and RMSE values.

Fig. 7 – *Coverage and RMSE changes graphed for changing neighborhood size*



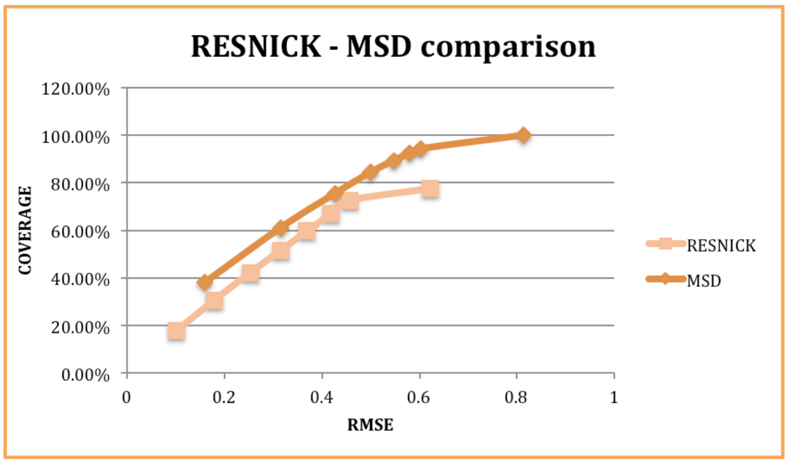
4 Discussion

Taking a moment to examine figure 8, it is abundantly clear that Resnick outperforms MSD in terms of RMSE. It is also clear to see that MSD has better coverage. Both collaborative filtering systems offer something different to the other. I think it is fair to say that Resnick/PCC is by far the better of the 3 collaborative filtering approaches I examined today.

By considering rating trends and patterns it has an edge over MSD and can create more accurate neighborhoods with a far smaller degree of error. It is interesting to mention that for the purposes of giving fair representations of data in figures 3, 4 and 5 I set the neighborhood to 200 so we could improve coverage as much as possible. When I did this, Resnick once again outperformed MSD and MAR giving me an RMSE value of 0.614964 compared to MSD's 0.733332 and MAR's 0.814535.

In conclusion Resnick is by far the superior algorithm and although it runs slower than the others steps can easily be taken to improve the runtime efficiency.

Fig. 8 *– Coverage and RMSE changes graphed for changing neighborhood size*



5 Conclusions

My RMSE, coverage and runtime varied greatly from algorithm to algorithm.

MAR had the poorest and least exciting values, but this was expected as the algorithm is ridiculously simplistic and involves only a single method. It is called statically and involved only simple addition, multiplication and division and does not use any more complex operations such as Math.sqrt / Math.pow / Math.abs which computationally expensive. It only contains 1 for loop and only primitive data types. The class itself is in total 23 lines long.

MSD is kind of the logical next step after MAR and before Resnick. It incorporates basic neighborhood construction and similarity computation and gives us a blueprint on which to build future more complex neighborhood collaborative filtering algorithms. Resnick's techniques takes the layout of my MSD class and improves it adding ever increasing constraints on neighborhood construction.

Without a doubt Resnick's technique was the best collaborative filtering technique we examined. It is the most interesting as it considers more factors when building its neighborhood than the more simplistic MSD technique. Although this approach has lower coverage at the same neighborhood threshold as MSD but as we increase the threshold we can obtain better coverage and better MSD.

6 References

[1]=http://docserv.uniduesseldorf.de/servlets/DerivateServlet/Derivate41263/ExpertRecommendation_Heck-3.pdf
[2] = https://www.cs.cmu.edu/~schneide/tut5/node42.html]
[3] = http://essay.utwente.nl/59711/1/MA_thesis_J_de_Wit.pdf
[4] = http://www.ijetae.com/files/Volume2Issue11/IJETAE_1112_59.pdf