Week 3 – Distance-Based Similarity

Filip Piskor – 12331436

Task

This week we were tasked with making predicitions based on the similiarity of users by calculating the distance between them (how alike are the users).

Implementation

I started off by creating a class called DistanceSimilarity and creating a method to calculate the distance between 2 users by checking the difference in the ratings of items they share. This difference was squared to obtain a positive answer and then averaged over all the items they did share. This method returns a Distance object which contains both the distance between 2 users and the amount of items they both share in common (corated).

Next I implemented a method to findNeighbours method which adds to the users neighbourhood based on the parameters specified. The two parametrs I chose to use is minimum of corated items required to qualify as a neighbour and size of the neighbourhood allowed. I thought these 2 parameters would help to achieve best neighbourhoods. I found the neighbours by creating an ArrayList and inserting each user that met the criteria into the list. While doing this I made sure both the user and the neighbour updated their information on the distance between them. This way I could reuse the information instead of calculating it again. This sped up my neighbourhood creation significantly. I then sorted the list and looped through the best ones until I reached the cut off point specified by the parameter size provided.

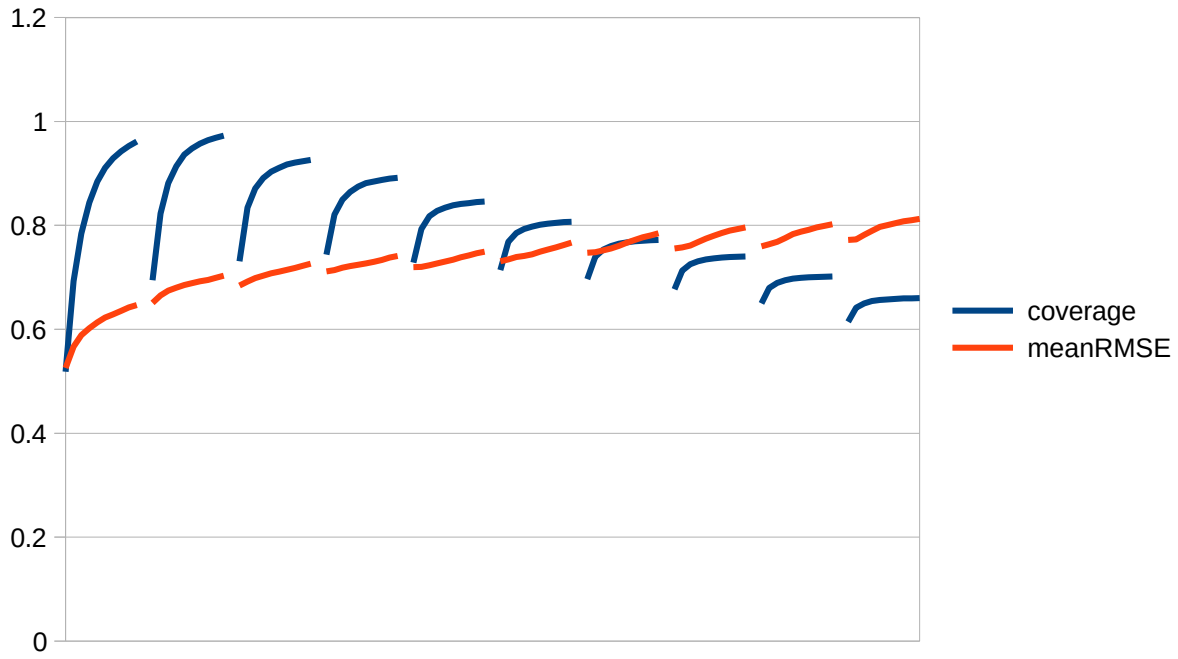
To implement the prediction method I used the formula provided in the slides. I also reused the distances calculated while adding neighbours which again helped to speed up my program significantly. The prediction takes the distance and between 2 users and makes it into a similarity which ranges between 1 and 0. This is then used as a weight to calculate how likely the user may have rated as their neighbour.

Next I implemented a test method which was basically a helper method that enabled me to run predicitions on all users and their items based on parameters that are specified. I used this method in the main test to try different paramaters of minimum corated items and size. It got the overall coverage and root mean square error for the given parameter.

Findings

I have tested the prediction method by starting off with 10 minimum corated items and 10 size and incrementing them by 10 each until both reach a 100. The results of this can be seen in the file “dist_sim_100.csv”. The results were quite surprising to me. I found that a neighbourhood of at least 10 corated items keeps the RMSE low and with greater neighbourhhod size its able to provide

great coverage. It also surprisingly provides a great coverage of users as the neighbourhood size gets bigger. We can see this in the graph below.



The above graph shows the coverage and mean root mean square error for each minCorated(10 to 100 in increments of 10) going from size 10 to 100. We see that the coverage fluctuates significantly when minCorated is low. It provides the lowest and the best coverage when size is changed. The coverage declines when minCorated is 30 and onwards and the fluctuation lowers. The RMSE seems to be quite linear as the minCorated increases while the coverage is rises exponentially.

Conclusion

Overall I found this experiment to be very insightful and I was pleasantly surprised when studying the data. This was one of the more interesting tasks to-date.