CSE 427 final project report
Netflix user rating prediction with collaborative filtering

Dina Elreedy, Chen Liu, Hang Yan, Hao Yan

December 13, 2016

1 Introduction

In this project we tackled the problem of predicting the ratings for movies, which can be helpful for recommender systems by recommending movies of high predicted ratings by a user to him. Formally, given a set of n users and k movies and an incomplete rating matrix U of n by k entries, our task is to predict the missing entries of U. The prediction is made possible with a large set of training data, consisting the incomplete rating of other m users for the same set of k movies.

There are multiple ways to solve this problem. The algorithm we are using is collaborative filtering, which takes the idea from kth nearest neighbor search (KNN). Given a user and the movie whose rating we want to predict, we first find similar users in the training set that have the ratings on this movie, then predict the unknown rating from these known ratings. The core of the algorithm is to find similar users with the testing user in the training set. Given the scale of the problem, we implemented our algorithm with MapReduce and execute it on the real cloud platform.

2 Motivation

Recommender Systems are used in diverse applications such as: Amazon or online markets for recommending products to customer that they may be interested in buying, social networks in form of recommending friends or pages, recommending restaurants or hotels and recommending movies for users to watch by Netflix, which is our project application.

Recommender systems are significant since companies can improve their sales using accurate recommender systems. Additionally, using recommender systems enhances customer satisfaction especially when he/she feels that the recommendations made by the system match their taste or needs to a great extent. Accordingly, both customers and companies would got some benefit from using recommender systems.

This project works on a subset of Netflix challenge dataset, which is a real dataset for a big data application. So we were interested in applying big data tools such as: Spark for preprocessing data analysis and Java Map Reduce for the main algorithm on a real cluster environment to solve such a real interesting big data application.

	Movie	User
Training set	1821	28978
Testing set	1701	27555

Table 1: The size of sampled data for analysis

3 Data analysis

The data we use is a subset of Netflix Prize data. The data consists of ratings for 17770 movies. There are ratings from 3255352 users in the training set and 100478 users in the tests set (data computed from the line count of TrainingRatings.txt, TestingRatings.txt and movie_titles.txt). The size of the problem indicates that cloud computing is necessary.

For a better algorithm design, we first performed analysis to the dataset with Spark. A random subset of the data is drawn for the analysis, see Table 1 for the size of sampled data. The analysis gives us the answer of the following problems:

Similarity measurement

The first key design choice is which similarity measurement to use for the *kth* nearest neighbor search. To this end, we compute the histogram of average user ratings of the training set, see Figure 1.

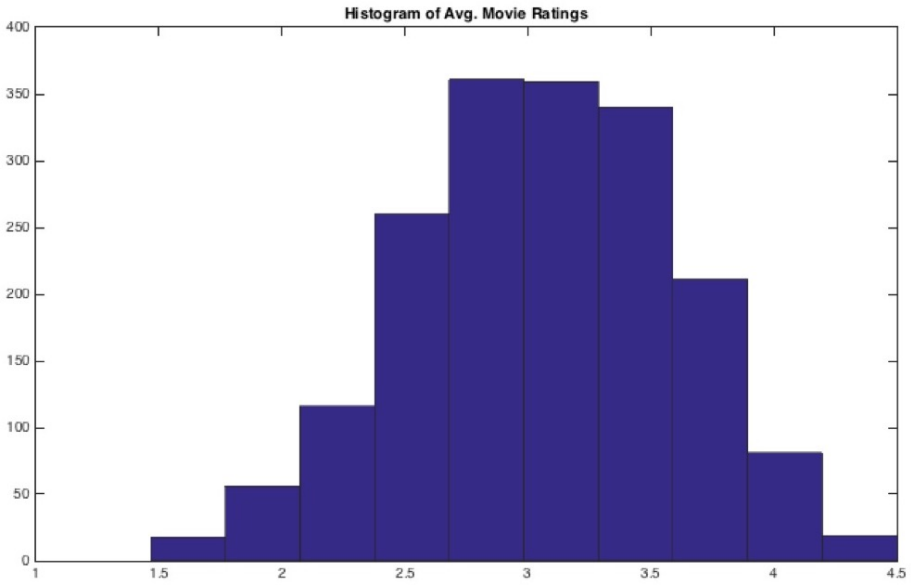

Figure 1: Histogram of average user ratings in the training set

The result suggests that most users rate normally. Therefore, we choose to use cosine similarity rather than Pearson similarity for efficiency.

User-user similiary vs. movie-movie similarity

To fill the missing entry, we can either use user as the key and fill with ratings of other similar users, or we can use movie as the key and fill with ratings of other similar movies. The choice depends of the expected overlay of users in test and train and items in test and train. See Table 2.

user-user	movie-movie
0.125467	0.439962

Table 2: Expected overly of two different keys

The result suggests that use movie-movie similarity is more appropriate.

Memory consumption

The algorithm will use two MapReduce jobs: the first job computes the list of users that rate each movie, and the second job computes the similarity of each pair of movies. The memory bottleneck happens at the reducer of each job. For first job, the expected memory usage is 111.5589MB and for the second job, the expected memory usage is 4.4MB.

To summarize, we decided to implement an algorithm that first computes movies that have similar user ratings with the query entry, then predicts the rating of the entry with known ratings from the similar movie. The similarity is measured by Cosine similarity.

4 Implementation

Our algorithm mainly consists of two stages: the first stage is finding the most similar movies for each movie in the test set based on their overlapping ratings and the second stage is predicting the corresponding missing entries of the ratings matrix U for pairs in the test set.

For the first stage, we have implemented it using three map reduce jobs:

- Job1: This job works on the training set, it produces movie-movie pairs and their ratings by each user.
  Mapper Input:($movie_i$,$user_k$,$rate_{ik}$) triplets
  Mapper Output:($user_k$,($movie_i$,$rate_ik$))
  Reducer Input:($user_k$,[($movie_1$,$rate_1k$),..($movie_i$,$rate_ik$),...]), all movies rated by user k along with their rates
  Reducer Output:($movie_i$,$movie_j$),($rate_{jk}$,$rate_{jk}$), all pairs of movies rated by user k and their rates

- Job2: This job works on the first job's output, the job mainly calculates cosine similarity between every pair of movies provided in the first job's output using their overlapped ratings.
  Mapper Input:($movie_i$,$movie_j$),($rate_{ik}$,$rate_{jk}$), rated pairs by user k.
  Mapper Output: ($movie_i$,$movie_j$),($rate_{ik}$,$rate_{jk}$) (It is Identity Mapper)
  Reducer Input:($movie_i$,$movie_j$),[($rate_{i1}$,$rate_{j1}$),($rate_{i2}$,$rate_{j2}$),...($rate_{ik}$,$rate_{jk}$)])
  Reducer Output:($movie_i$,$movie_j$),$cos_sim(movie_i, movie_j)$

- Job3: After calculating similarity between each movie pairs, the third job selects the top-k similar movies for each movie. The default K we use is 10, however, we can pass K as an input parameter using command line.
  Mapper Input:$((movie_i,movie_j),cos_sim(movie_i,movie_j)$
  Mapper Output:$(movie_i,(movie_j,cos_sim(movie_i,movie_j))$
  Reducer Input:$(movie_i,[(movie_j,cos_sim(movie_i,movie_j),.(movie_k,cos_sim(movie_i,movie_k),....])$,
  list of all overlapped movies with $movie_i$and their corresponding rates
  Reducer Output:Top-K similar movies per movie i: $(movie_i,[(movie_j,cos_sim(movie_i,movie_j),.(movie_k,cos_sim(m$

For the second stage, since the top-K entries per movie is a small number, then this task does not require a Map Reduce job. Accordingly, we have implemented it using....

5 Cloud Execution

The size of the problem exceeds the computation resources of a single virtual machine. We therefore use the HDInsight service on Microsoft Azure for the full execution.

We created a cluster with 2 master nodes and 4 computing nodes. Each of the computing nodes has the configuration of 4 cores CPU, 14GB RAM and 200GB local disk. See Figure 2.

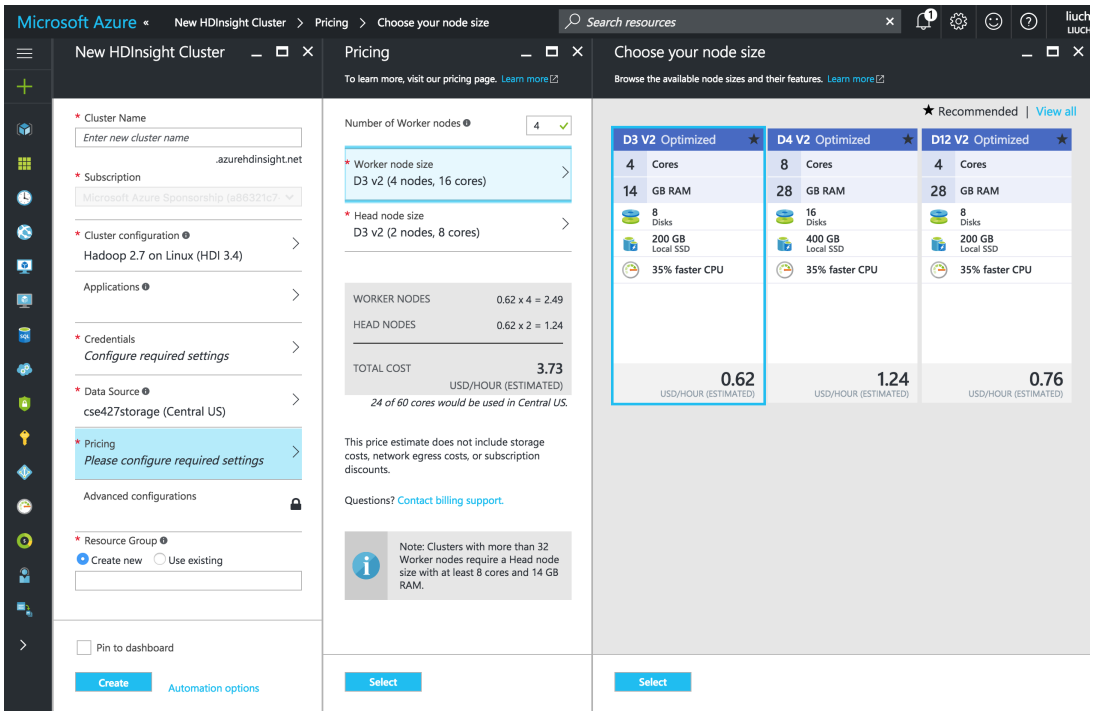

Figure 2: Node configuration

The computing time with the cluster is listed in Table 3.

Job	Job 1 (item-item pair)	Job 2 (similarity)	Job 3 (top list)
CPU Time(s)	279.55	1516.17	13.51

Table 3: CPU time on the cluster

6 Results

In this section, we show results of our experiments on the netflix subset dataset. We evaluate our algorithm performance according to the prediction accuracy (using root mean square eror(RMSE) and mean absolute error(MAE) and the number of unpredicted ratings in the test set.

One significant parameter in our algorithm is K, number of similar movies to consider in calculating ratings' predictions. We have tried different settings of K to investigate how it can affect the performance of our algorithm.

Figure 3 and Figure 4 shows the RMSE and MAE for our collaborative filtering algorithm using different values of the number of considered similar movies (K). The error measures, RMSE and MAE, are computed for all entries the test set, and for the entries that we could not predict, we use the average rating for the movie under test as our prediction. The average rating per movie is calculated by a pre-processing Map Reduce job.

We compare different settings for prediction in Figure 3 and Figure 4 including: using average rating prediction of the top K movies, using weighted average prediction of the top K movies, and using rounded versions for both average prediction and weighted average prediction. Additionally, we compare our results to two baselines: the first is predicting the global mean of movie ratings for all entries, while the second baseline uses the mean rating of the movie which is being predicted. These mean estimates are calculated using the training ratings only.

It can be observed from Figure 3 and Figure 4 that increasing number of considered similar movies K helps increasing accuracy which is reasonable since increasing K enables our algorithm to predict more missing entries, and accordingly accuracy for these entries increases over the default movie mean rating we have for unpredicted entries. These figures show that using weighted average predictions is slightly better than using average predictions. However, this improvement is slight since we have observed that the variance of cosine similarities between the top K movies being rated by the same user under test is not large. Regarding, rounding the predictions, it can be noticed that rounding reduces MAE while it largely increases RMSE over the movie mean baseline as shown in Figure 3 since RMSE is more sensitive than MAE to outliers, so if rounding goes on the wrong direction, RMSE would increase. Accordingly, we have used unrounded version of weighted average prediction for further experiments.

Figure 5 shows the percentage of unpredicted ratings using our collaborative filtering approach as we varies K.It can be observed from this figure that increasing K has a significant impact on reducing percentage of unpredicted ratings we have, this is reasonable as considering more similar neighbours (movies), would increase the probability of having some of these movies rated by the users in the test set, and accordingly we can provide more predictions to the missing entries in the test set. However, increasing K beyond some extent would not help much decrease the number of unpredicted entries. Moreover, it could hurt accuracy since we are considering many distant (non-similar) movies if we set K greater than 500 as shown in Figure 3 and Figure 4 that errors start to saturate or even drop for the MAE at this point. For $K = 500$, the percentage of unpredicted

ratings is 0.84% and for $K = 1000$, the percentage of unpredicted ratings is 0.824%.

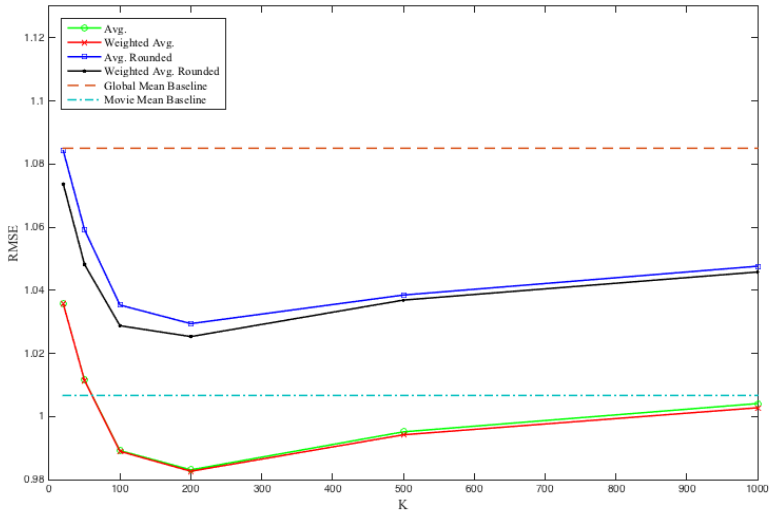

Figure 3: RMSE for our collaborative filtering approach using different K values

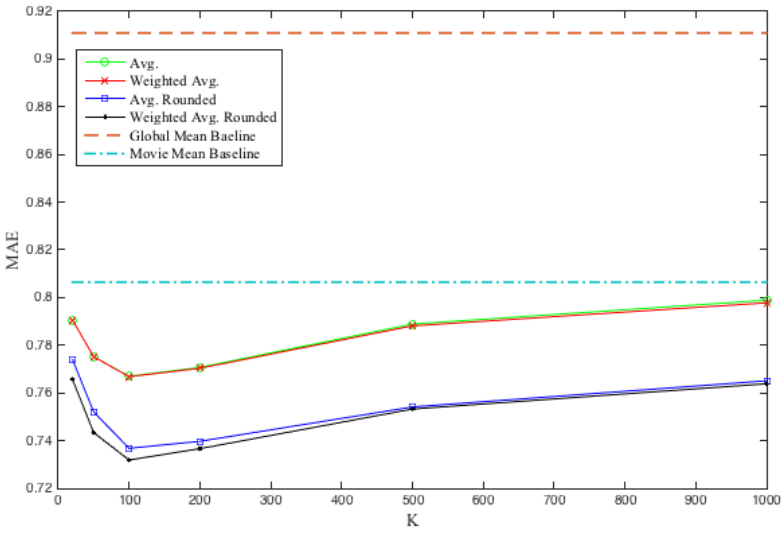

Figure 4: MAE for our collaborative filtering approach using different K values

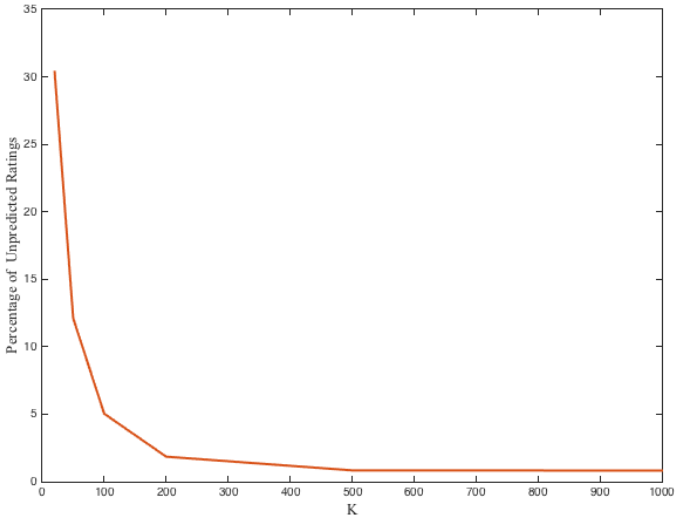

Figure 5: Percentage of unpredicted ratings using different K values

According to our results, increasing K helps improving the overall accuracy, however, for those we can rate using small K's, increasing K would reducer their accuracy since we are considering more distant movies. This can be observed from Figure 6 which shows the RMSE for the predicted entries only in the test set, excluding the ones that we could not predict using current K.

From the results above, we have developed an adaptive prediction scheme that tries to handle both issues of predicting nearly all the missing ratings in the test set, which implies using a large K. However, our scheme aims at not sacrificing the accuracy of the already predicted entries , the ones having many overlapping movies that we can predict their ratings with using smaller K. Our scheme is to use $K = 500$ as it is the best point in terms of both accuracy and percentage of missing ratings 0.84%, Figure 3 and Figure 4. However, we limit the number of neighbours (similar movies) actually used in rating prediction to the top M out of these K movies, where M is much less than K.

Figure 7 and 8 show RMSE and MAE for this setting using weighted average predictions, $K = 500$ and with variable M from 5 to 100. It can be observed that having too small $M < 20$ does not yield the best performance, we think the reason for that can be that some of considered K movies may have only one common user rating them, having 1 cosine similarity, however, such movies with low number of overlapped ratings may not be enough to have accurate predictions. On the other hand increasing M too much would increase the error, this is reasonable and this is the motivation for using M. Accordingly, we pick $M = 20$, for our final predictions, which was the best point for both Figure 7 and 8.

Table 4 summarizes our final results for our developed collaborative filtering approach using $K = 500$ and $M = 20$. Results are presented in terms of prediction accuracy using RMSE and MAE, and for the number and percentage of predicted ratings that our algorithm was able to make. The table also includes the RMSE and MAE for the two baselines we compare to, the global mean

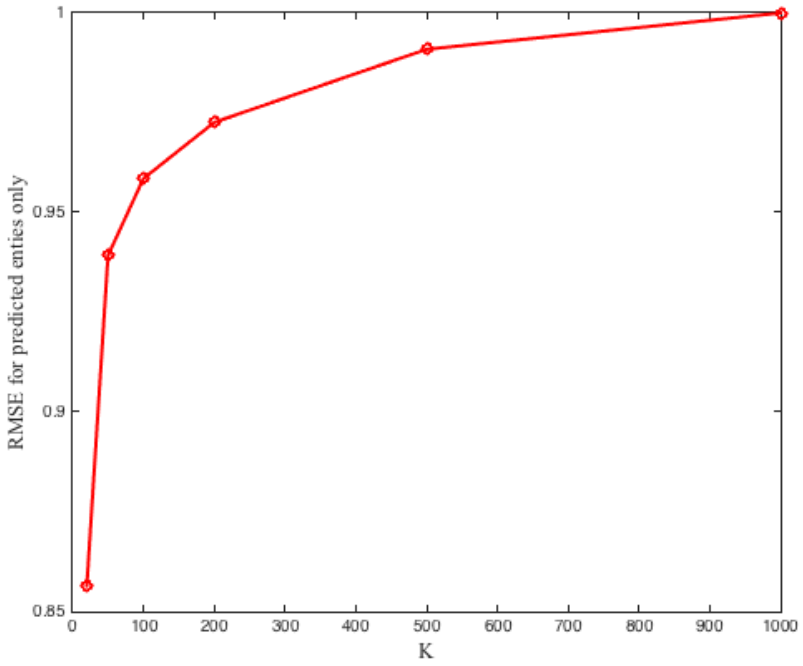

Figure 6: RMSE using Weighted Avg. Prediction for predicted entries only using different K values

rating and movie mean rating. Table 4 shows that our collaborative filtering approach have better accuracy than the two baselines and we almost predict every missing entry.

Method	RMSE	MAE	Number of Unpredicted Ratings	Percentage of Unpredicted Ratings
Collaborative Filtering	**0.98099**	**0.76806**	841	0.84%
Global Mean Baseline	1.08502	0.91094	-	-
Movie Mean Baseline	1.0067	0.8065	-	-

Table 4: Summary of our collaborative filtering and two baselines prediction results

7 Conclusion

Appendix A: group collaboration

Our group consists of four members. The tasks for each group member is listed below:

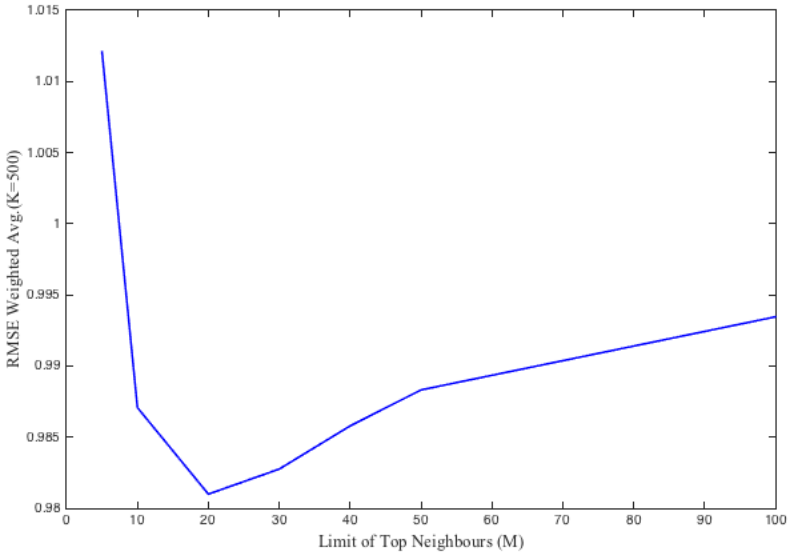

Figure 7: RMSE using Weighted Avg. Prediction ($K = 500$) using different M values

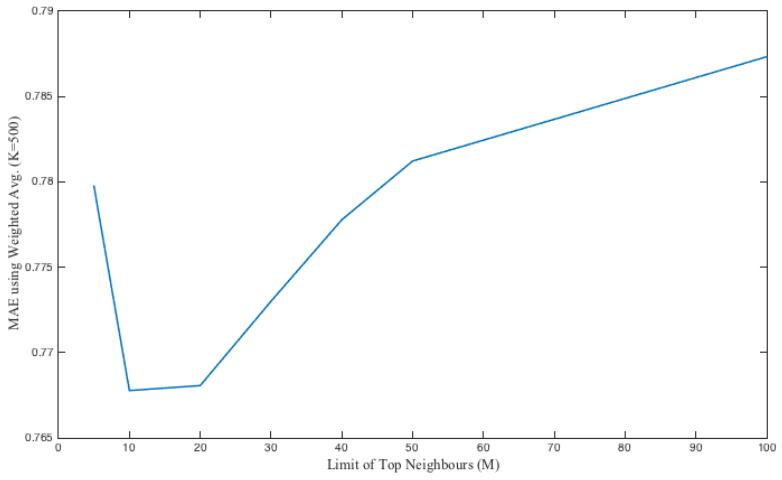

Figure 8: MAE using Weighted Avg. Prediction ($K = 500$) using different M values

Name	Task
Dine Elreedy	Algorithm design, collaboarative filtering, report
Chen Liu	Algorithm design, collaboarative filtering, prediction
Hang Yan	Algorithm design, cloud platform, report
Hao Yan	Algorithm design, data analysis

Table 5: Tasks for group members