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

Dina Elreedy, Chen Liu, Hang Yan, Hao Yan

December 14, 2016

1 Introduction and motivation

Recommender Systems are used in diverse applications. Online markets recommend products to customer that they may be interested in buying; social networks recommend friends or pages; online streaming services (e.g. Netflix) recommand movies to users.

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.

In this project we tackled the problem of predicting the ratings for movies from a set of users, which will guide the recommender system to recommend movies that matches his/her interest. Formally, given a set of n users and k movies and an incomplete rating matrix U of n by k entries, where U_{ij} represents the rating from user i to the movie j, 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/movies 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/movies with the testing user/movie in the training set. Given the scale of the problem, we implemented our algorithm with Spark and MapReduce and execute it on the real cloud platform.

2 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 (numbers 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:

	Movie	User
Training set	1821	28978
Testing set	1701	27555

Table 1: The size of sampled data for analysis

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 in the training set, see Figure 1.

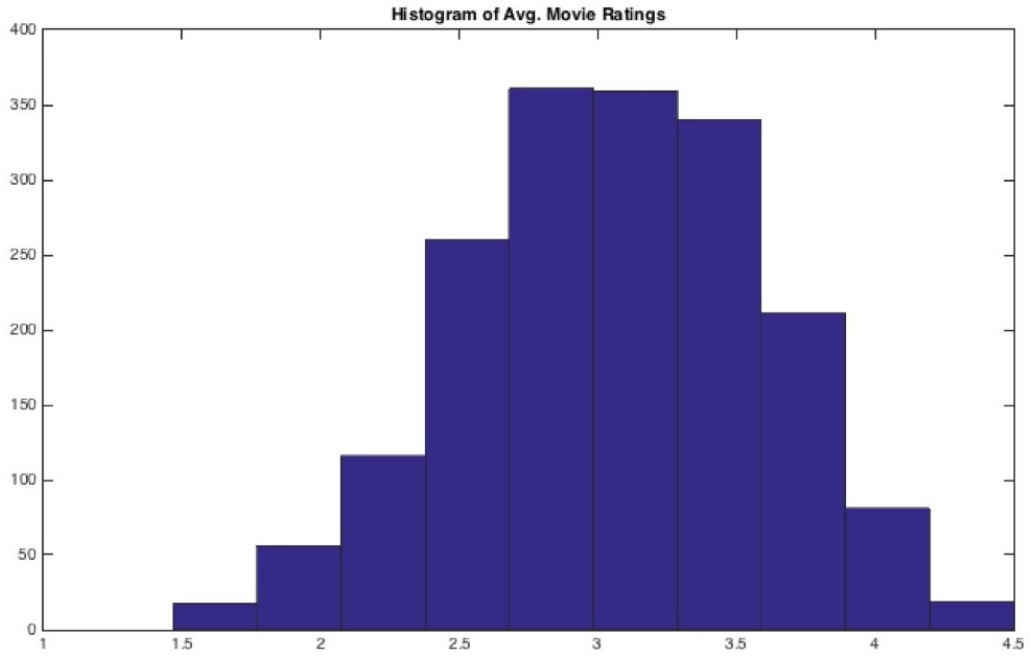

Figure 1: Histogram of average user ratings in the training set

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.

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

user-user	movie-movie
0.125467	0.439962

Table 2: Expected overly of two different keys

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, given a user with missing ratings, we predict those missing ratings one by one. For each movie whose rating is missing (the query movie), the algorithm first computes movies that have similar user ratings with the query movie, then predicts the missing rating with known ratings from the similar movies. The similarity is measured by Cosine similarity.

3 Implementation

Our algorithm consists of two major 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	($movie_i$,$movie_j$),$cos(movie_i, movie_j)$

- 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	$((movie_i,movie_j),cos(movie_i, movie_j)$
Mapper Output	$(movie_i,(movie_j, cos(movie_i, movie_j))$
Reducer Input	$(movie_i, [(movie_j, cos(movie_i, movie_j)) \ldots])$, list of all overlapped movies with $movie_i$and their corresponding rates
Reducer Output	$(movie_i, [(movie_j, cos(movie_i, movie_j)) \ldots])$, Top-K similar movies per movie

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

4 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.

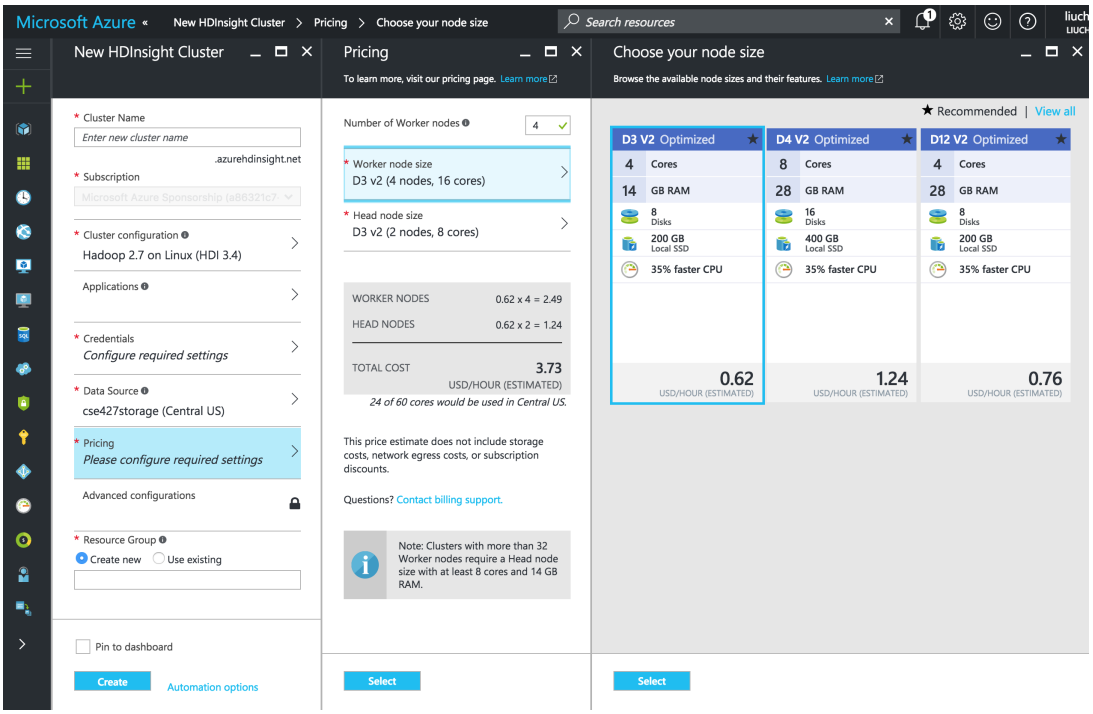

Figure 2: Node configuration

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

5 Results

In this section, we show results of our experiments on the netflix subset dataset. We evaluate our algorithm performance by 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 important parameter in our algorithm is K, the 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 in the test set. For the entries that we could not predict (due to the small K), we use their average ratings in the training set 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, and the second baseline uses the mean rating of the movie which is being predicted. These mean estimates are calculated using the training ratings only.

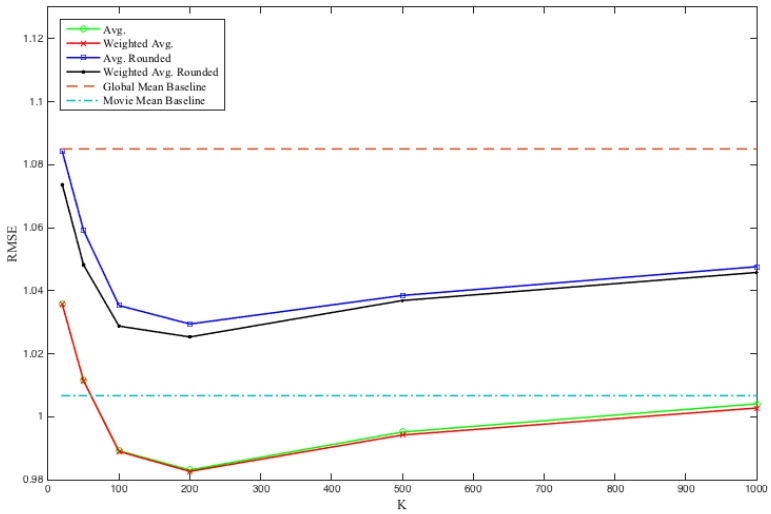

Figure 3: RMSE for our collaborative filtering approach using different K values

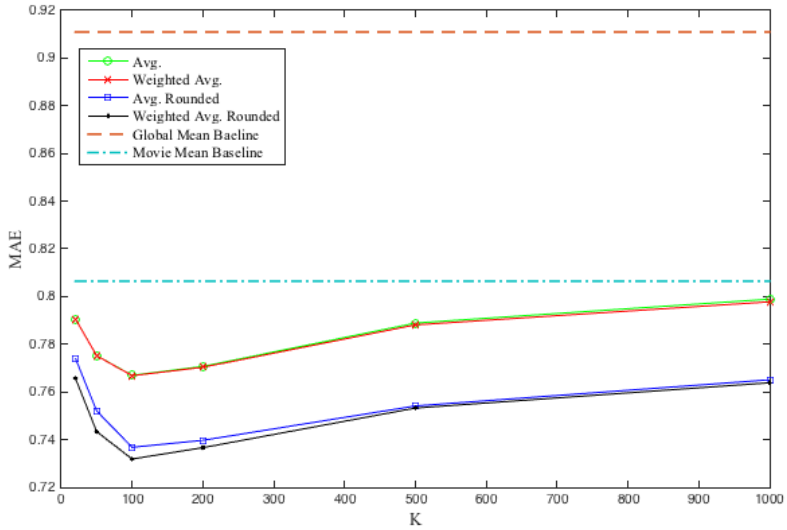

Figure 4: MAE for our collaborative filtering approach using different K values

It can be observed from Figure 3 and Figure 4 that increasing number of considered similar movies K helps increasing accuracy. This can by expained by the fact that increasing K enables our algorithm to predict more missing entries, which gives higher accuracy for these entries. These figures show that using weighted average predictions is slightly better than using average predictions. However, this improvement is insignificant since 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. If rounding goes on the wrong direction, RMSE would increase. Therefore, 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%.

According to our results, increasing K below a threshold helps improving the overall accuracy. However, increasing the K above that threshold will descrese the accuracy. This can be observed from Figure 6 which shows the RMSE for the predicted entries in the test set, excluding the ones that we could not predict using current K. This observation agrees with the theory in machine

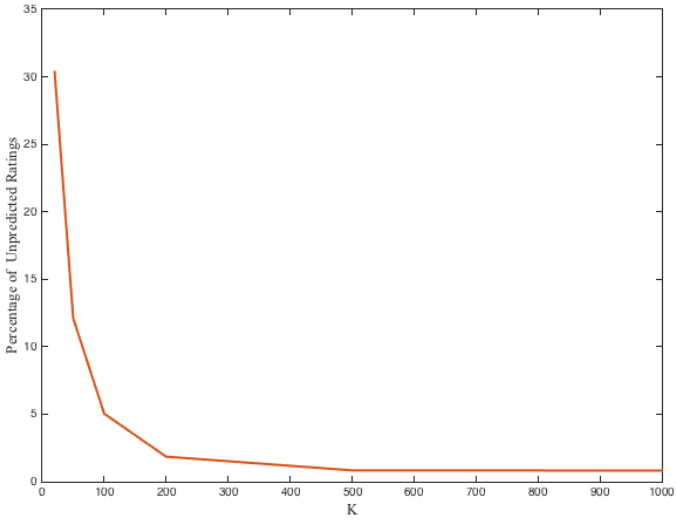

Figure 5: Percentage of unpredicted ratings using different K values

learning, which states that increasing K will reduce the bias of the model, but will increase its variance. We have found that the "sweet point" of K is around 200.

To balance the ratio of predicted entries and the overal accuracy, we developed an adaptive prediction scheme. We use two parameters to control the whole process: for a specific user i, and movie j, we first find $K = 500$ candidate movies using the above algorithm. In the next step, instead of using all 500 similar movies, we only use top M movies that are rated by the user i. In the case that the user i rated less than M movies among all $K = 500$ candidates, we drop parameter M and use all K candidates.

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. With $M = 20$ we are able to have comparable accuracy with $K = 200$, while the comparable amount of predicted entries with $K = 500$, See Table 4.

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 percentage of unpredicted ratings. The table also shows the comparison against two baselines methods: the global mean rating and movie mean rating. Our collaborative filtering approach have better accuracy than the two baselines with low percentage of unpredicted entries.

	RMSE	MAE	Percentage of unpredicted ratings
Collaborative Filtering	**0.98099**	**0.76806**	0.84%
Global Mean Baseline	1.08502	0.91094	-
Movie Mean Baseline	1.0067	0.8065	-

Table 4: Result of our collaborative filtering and the comparison against two baseline methods.

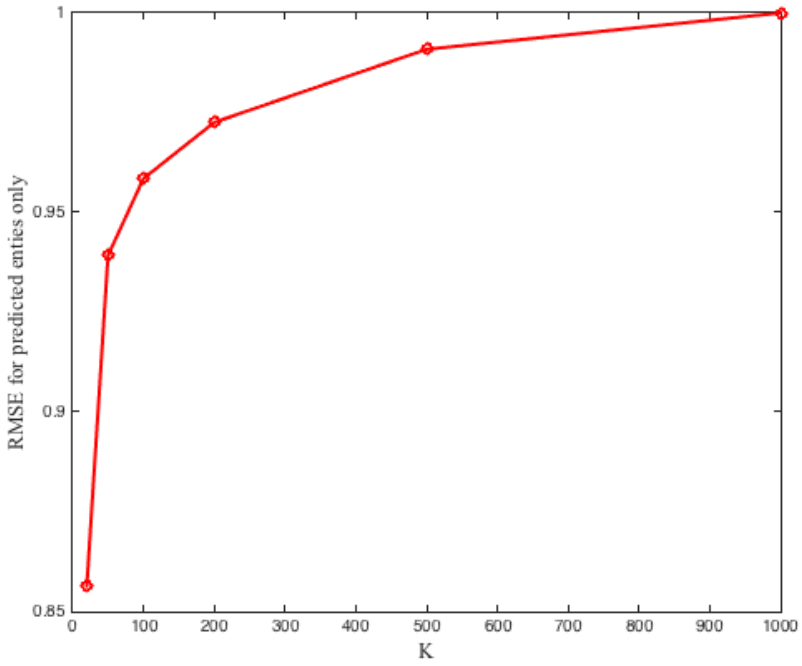

Figure 6: RMSE using Weighted Avg. Prediction for predicted entries only using different K values

6 Conclusion

In this project we have developed a system that predict missing ratings of a movie from a user by collaboarative filtering. We analyzed the data by Spark and designed the algorithm based on the analysis. The collaboarative filtering algorithm is implemented with MapReduce framework and executed on Microsift Azure Hdinsight service. We have also improved the origin collaboarative filtering and predicting algorithm to balance the prediction accuracy and number of predicted entries. Experimental results demonstrate the effectiveness of our system and show the impact of several parameters.

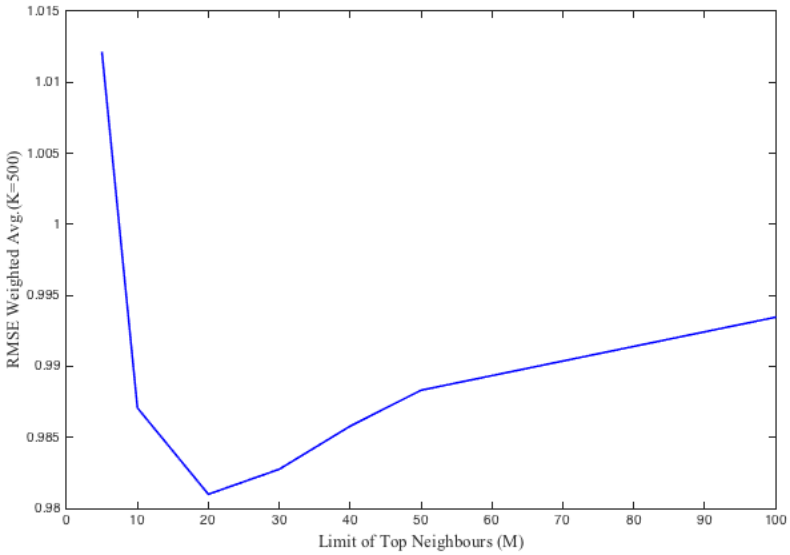

Figure 7: RMSE with varying M using weighted average prediction and $K = 500$.

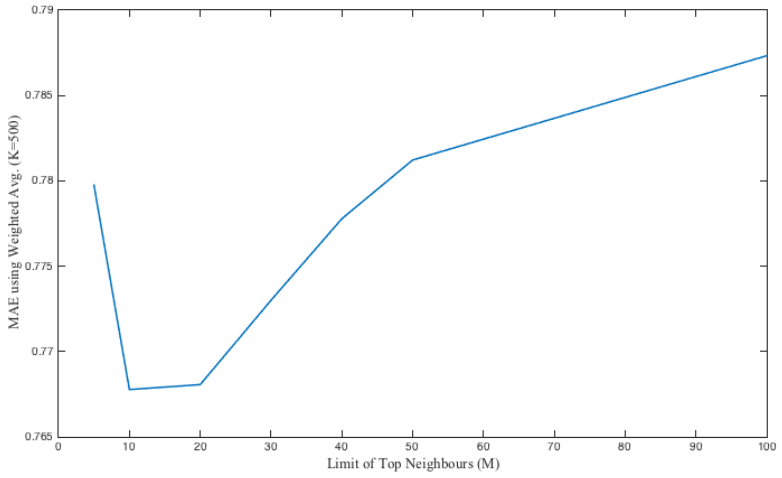

Figure 8: MAE with varying M using weighted average prediction and $K = 500$.

Appendix A: group collaboration

Our group consists of four members. The tasks for each group member is listed in Table 5:

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