Collaborative Filtering

Leonard Dervishi -21600316
Anisa Llaveshi - 21300392
Figali Taho - 21403059
Syed Sarjeel Yusuf -21402744

March 16, 2017

1 Introduction

In this project we build a recommender engine by implementing collaborative filtering algorithms to make predictions on the ratings that users will give to items and to make recommendations to the users accordingly. We implement three different algorithms: item-item, user-user and latent factor algorithms. We analyze each of these algorithms and calculate the error rate in the predictions using Root Mean Squared Error as an error metric. Lastly, we compare the accuracy of these algorithms with each other and analyze on the advantages and disadvantages of each.

2 Datasets

In this project we use three different datasets from the MovieLens website. The first dataset consists on 100,000 ratings from 1000 users on 1700 movies. The second dataset consists on 1 million ratings from 6000 users on 4000 movies. The third dataset consists on 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users[1]. Each of the datasets is divided in 5 subsets of 80%20% data in disjoint training and test sets. We use the 80% data sets to train the algorithm and the 20% data set as unseen data to test the algorithms. We run 5-fold cross validation using these subsets on each of the algorithms.

3 User - User

User user collaborative filtering is a methodology which makes predictions on the rating that a user will give an item, based on ratings done by similar other users. To calculate this similarity between users we have used two different similarity metrics: Cosine Similarity and Pearson Correlation. In our project we have also implemented the biased user-user methodology.

3.1 Cosine Similarity

As a definition cosine similarity measures the angle between two non-zero vectors in space. In this case it is used to calculate how close are two different users based on their ratings on items. The formula to calculate cosine similarity is given below.

$$sim(x,y) = \frac{\sum_{i=0}^{n-1} x_i y_i}{\sqrt{\sum_{i=0}^{n-1}(x_i)^2}\sqrt{\sum_{i=0}^{n-1}(y_i)^2}}$$

For the implementation of user-user collaborative filtering, the first step is the conversion of Movielens dataset to a dictionary which is in the following format `data={user:{movie:rating}}`.

The next step is creating another dictionary which will hold the similarity values between all users in the format: `sim={user:{user:simValue}}`. In the implementation there is a function which takes as a parameter a user and returns a sorted list of the most similar users to that one. Another function calculates the ratings of a certain movie for the selected user based on the ratings of that movie by top k similar users (neighbors). Generally the value of k depends on the dataset. The algorithm is tested with different k and in the end the ideal k is taken as the average of the test results.

To check the effectiveness of the algorithm we have used RMSE.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n} y_j - \overline{y_j}}$$

Initially we use the 100K dataset. The graph shown in Figure 1 below is the average of the RMSE values taken from all the 5 test sets.

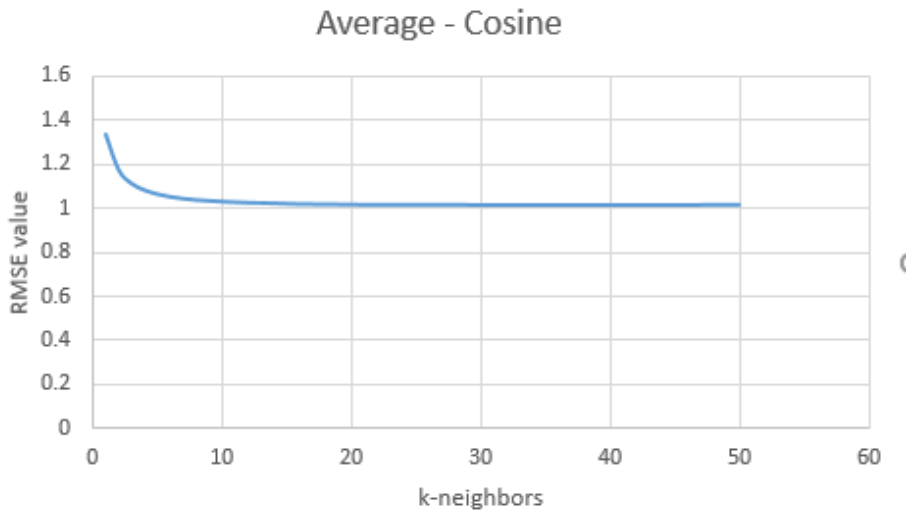

Figure 1: Lowest RMSE value at k = 38. RMSE = 1.012217.

From the figure we can understand that as k increases the value of RMSE

decreases. But the decrease stops at some point since the algorithm starts taking in consideration even users who are not very similar with the selected one.

3.2 Pearson Correlation

The calculations done in Pearson Correlation methodology are similar to the one mentioned above. However, the calculation of the similarity between users is different since here it is used data normalization. Each rating of a movie is subtracted by the average rating of that user. The formula for Pearson Correlation is given below.

$$sim(x,y) = \frac{\sum_{i=0}^{n-1}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=0}^{n-1}(x_i - \overline{x})^2}\sqrt{\sum_{i=0}^{n-1}(y_i - \overline{y})^2}}$$

The same testing as explained in section **3.1** is done with the algorithm of Pearson Correlation and the results are displayed in Figure 2.

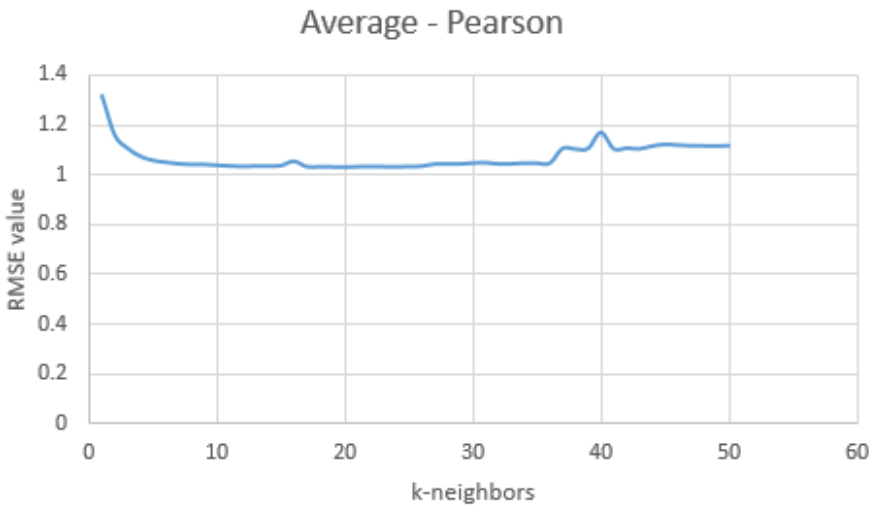

Figure 2: Lowest RMSE value at k = 20. RMSE = 1.029401.

Since the similarity between users is more accurate in Pearson Correlation we see that taking in consideration too many similar users is bad. The reason is that only a few of them are really similar and the others will just affect badly in the result and increase RMSE value. This is also the reason that we got a lower value of k as an average.
Pearson Correlation is also tested with 1M dataset. The results for 5 differently split up dataset are shown in Table 1. (K=20).

1M dataset is tested with only K = 20 since it needed much more time compared with 100K dataset. From the results that were obtained with the 1M dataset it is understood that more data accounts for more accurate predictions compared to 100K.

Test Set	RMSE
U1	0.8114768253
U2	0.8030312625
U3	0.7959279677
U4	0.8002061771
Average	0.8026605581

Table 1: RMSE values for different sets.

3.3 User - User with baseline

The biased user-user similarity implemented incorporated the Pearson correlation along with taking into account the baseline. The modification to calculate the rating is as below:

$$r_{xi} = b_{xi} + \frac{\sum_{y \in N} S_{xy} r_{yi}}{\sum_{y \in N} S_{xy}}$$

where:

$$b_x i = u + b_x + b_i,$$

$$b_x = u_x - u$$

$$bi = u_i - u$$

It can be seen that the baseline is the summation of the global average(u), the average rating of all the movies, with the deviation of the item i and the deviation in average rating of the user x. The algorithm was implemented using three datasets from 100K, each time changing the nearest neighbor by 5 from 10 to 35. This is better explained in figure. It must also be noted only datasets 1 and 3 yielded results whereas the other test datasets were problematic in terms of sparsity.

K-Neighbors	Set = 1	Set = 3	Avg
10	1.061915	0.9234706	0.9926928
15	1.064152	0.923852	1.03617555
20	1.1807644	0.924644	0.9915705
25	1.058481	1.042215	1.050348
30	1.024438	1.042248	1.033343
35	1.09535	1.0012559	1.04830295

Table 2: RMSE values for user-user with baseline

4 Item - Item

The similarity metric used for the weighted sum and weighted sum with baseline implementations is the adjusted cosine similarity, given in the formula below

$$sim(i,j) = \frac{\sum_{u \in U}(r_u, i - \bar{r_u})(r_u, j - \bar{r_u})}{\sqrt{\sum_{u \in U}(r_u, i - \bar{r_u})^2}\sqrt{\sum_{u \in U}(r_{u,j} - \bar{r_u})^2}}$$

where $r_{u,i}$ is the rating of user u on item i, and $\bar{r_u}$ is the average rating of user u. The purpose of this formula is to take into consideration personal ratings schemes

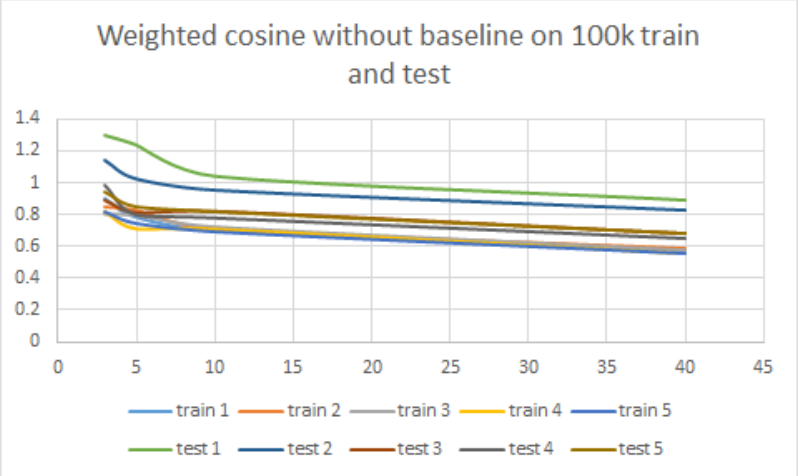

Figure 3: Change in error rate as k value increases for each of the 5 different test and train datasets of the 100k movies.

of each user. This is a good point to consider since some users might prefer to rate high in general, while others prefer the opposite.

4.1 Weighted sum with no baseline

The weighted sum formula for making predictions without a baseline is given below:

$$p(u,i) = \frac{\sum_{j \in N(i;u)} sim(i,j)(r_{u,j})}{\sum_{j \in N(i;u)} sim(i,j)}$$

This is the formulation we used to find predictions of ratings users u would give to item i without considering the baseline. The sum in the numerator considers all items j that user u has rated and that are also among the k most similar items to item i. The values of k were taken different and the algorithm implementing weighted sum in `python` was run among five test and training sets of the 100k movie dataset that MovieLens provided, and also on the 5 different test and training sets of million ratings dataset of MovieLens.

	K = 3	K = 5	K = 10	K = 40
Train RMSE	0.844144	0.822962	0.706956	0.585438
Test RMSE	1.140517	1.023821	0.952792	0.825943

Table 3: Weighted sum with no baseline, values for a train and test set with increasing K-nearest neighbors.

As it can be understood from the graph in figure 3, the tuning of the k value is important for the algorithm, as a higher k implies that there is a higher

accuracy in predictions.
The weighted cosine algorithm was also run for the 1 million ratings MovieLens dataset. Since the dataset didn't come equipped with a division of test and training data files, we implemented a script in python for dividing the data randomly into five files with 80% data used as the training sets and five files with 20% data used as test files. Additionally, we had another approach to dataset splitting where we split in 20% of the ratings data ordered by the user id. we obtained results for both. In table 4 the ordered splitting subsets are used. The below data was obtained for one of the training and test data files.

	K = 10	K = 35	K = 40
Train RMSE	0.944144	0.7606645	0.6492337
Test RMSE	0.965846	0.8431555	0.7421862

Table 4: RMSE values for 1 million dataset training and test datasets.

As for the smaller dataset of 100k, the large dataset of 1 million also has similar trends, with the values also being smaller, indicating of better results for larger datasets.

4.2 Weighted sum with baseline estimate

The weighted sum formula with added baseline is given below.

$$p(u,i) = b_{u,i} + \frac{\sum_{j \in N(i;u)} sim(i,j)(r_{u,j} - b_{u,j})}{\sum_{j \in N(i;u)} sim(i,j)}$$

The prediction of the rating user u will give to item i will be added a baseline estimate which is defined as:

$$b_{u,i} = \mu + b_u + b_i$$

where μ is the mean of all ratings in the dataset, b_x is the mean of the ratings of user u and b_i is the mean of all ratings of item i. The baseline weighted sum formula is also updated to consider the baseline of item j as well, with this baseline estimate being subtracted of rating of item j by user u. Below is some data obtained from k tuning on train and test datasets for the 100k dataset. The algorithm performs better than the previous algorithm with no baseline, while there is still the trend of having better results with bigger k values.

	K = 3	K = 5	K = 10	K = 40
Train RMSE	0.8199565	0.7306712	0.7228179	0.571253427
Test RMSE	0.978926227	0.787742891	0.721377677	0.641638559

Table 5: RMSE values for weighted sum with baseline on a test and training dataset for increasing K-nearest neighbors.

In a similar way as before, running this algorithm on the 1 million training and test datasets, gives the following results. These again indicate that the weighted cosine with baseline estimation and larger data performs better than the previous algorithms on average.

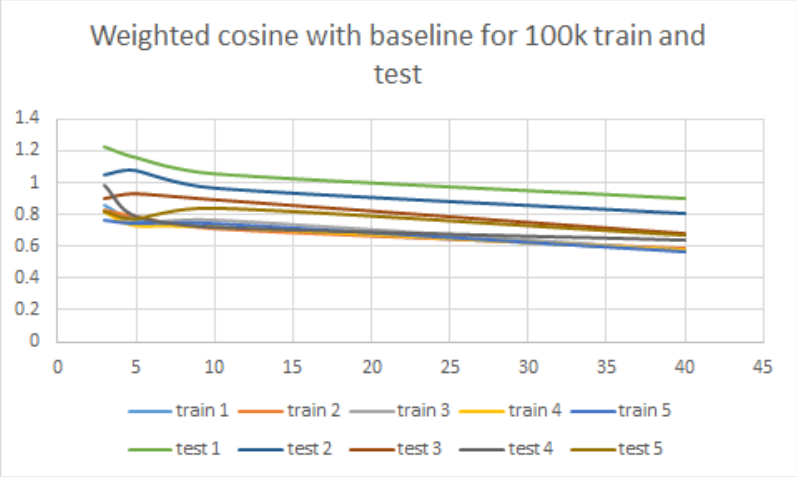

Figure 4: Change in error rate as k value increases in baseline weighted cosine for each of the 5 different test and train datasets of the 100k movies.

	K = 10	K = 35	K = 40
Train RMSE	0.9140256	0.7566674	0.5000142
Test RMSE	0.932849075	0.7710564	0.5124863

Table 6: RMSE values for 1 million dataset training and test datasets using weighted cosine with baseline.

A 5-fold cross validation for both datasets was performed.
On the 100k test and training datasets, the mean of the test error rates results to be 0.994384 on the weighted cosine without baseline and 0.973736 on the weighted cosine with baseline. On the 1 million data the mean rmse results to be 0.95729 for the weighted sum implementation without baseline and 0.944384. These results are obtained from k neighbor value of 3.

4.3 Slope one

The Slope One algorithm tries to predict the rating of a certain item, by a certain user, by considering the trends in ratings of the user as compared to all the other users. The basic constructs of the slope one algorithm can be divided into two parts.
The first part involves calculating the deviations in ratings that the specific user makes, as compared to all the other ratings made by the other users. The formula can be seen as below:

$$dev_{(i,j)} = \sum_{u \in S_{i,j}(X)} \frac{u_i - u_j}{card(S_{i,j}(X))}$$

The above formula first calculates deviations in ratings among pairs of items. It sums the difference between the ratings of item i and j. It is then divided by the cardinality of the pair. This is the total number of users that rated both items. In doing so one derives a deviation matrix between all the items. The second step is to calculate the ratings a user would give to a certain item depending on the the patterns of deviation among the item and all other items, and how the user has rated the other items. This is done using the formula below:

$$P^{wS1}(u)_j = \frac{\sum_{i \in S(u)=j} (dev_{j,i} + u_i) c_j, i}{\sum_{i \in S(u)=j} c_j, i}$$

The numerator sums the rating a user gives to a movie with the deviation value of the movie whose rating is being predicted paired with the movie that the user has given a rating for. The $c_{i,j}$ refers to the cardinality and as can be seen from the formula, it is multiplied with the resultant sum. The denominator is simply a summation of all the cardinalities of all the items that the user X has rated. The resultant value is thus the rating predicted (P).

After the algorithm was implemented it was tested with a dataset consisting of 100k entries. The algorithm was run with three different data sets as seen below:

Data Set:	Set = 1	Set = 2	Set = 3
Test RMSE	1.041421	1.00017	1.036723

Table 7: RMSE values for different subsets for Slope One

All RMSE values were under 1.1 . It also must be noted that multiple tests could only be done on the basis of changing the test data, and not specifically other factors. This is because, the Slope One algorithm, unlike other algorithms which work on similarity, does not rely on the nearest neighbors. Hence changing K-neighbors is not an option.

5 Latent Factor

In the latent factor model, given the ratings matrix we try to predict the missing entries in the matrix by factorizing the ratings matrix $R_{\text{user x item}}$ as the dot product of two matrices $Q_{\text{factor x item}}$ and $P_{\text{user x factor}}$ where factor is the number of latent factors (dimensionality) that we need to define. Our aim is to find these matrices P and Q such that we minimize the error rate between our prediction and the true ratings. Next we will explain two different methods for this matrix factorization.

5.1 Without baseline

Our goal is to find P and Q such that we minimize the following function:

$$\min_{P,Q} \sum_{(i,x) \in R} (r_{xi} - p_x \cdot q_i)^2$$

$$\hat{r_{xi}} = p_x \cdot q_i = \sum_f q_{if} \cdot p_{xf}$$

p_x is column x of matrix P^T and q_i is row i of matrix Q
In order to solve this minimization problem we need to define what the number of factors f is. However, the overfitting problem arises. If f is too large the model will fit too well the training data and will not generalize for unseen data. In order to solve the overfitting problem we introduce two new variables λ_1 and λ_2 and we apply regularization to the objective function that we aim to minimize:

$$\min_{P,Q} \sum_{(i,x)\in R} (r_{xi} - p_x \cdot q_i)^2 + [\lambda_1 \sum_x ||p_x||^2 + \lambda_2 \sum_i ||q_i||^2]$$

This new objective function allows rich model where there are sufficient data and shrinks aggressively where data are scarce.

Stochastic Gradient Descent

To minimize our objective function we use the Stochastic Gradient Descent Algorithm. In the beginning we initialize P and Q with random values. We run the SGD algorithm on a fixed number of loops. We compute the gradient and update the equation on each step for each rating.

For each rating r_{xi}:

$\epsilon_i = r_{xi} - p_x \cdot q_i$

$q_i \leftarrow q_i + \mu_1(\epsilon_{xi} p_x - \lambda_2 q_i)$

$p_x \leftarrow p_x + \mu_2(\epsilon_{xi} q_i - \lambda_1 p_x)$

5.2 With baseline

In this algorithm we do not assume that the rating of a user for an item is based only on the dot product of the respective latent vector but we take into consideration also the item and the user bias as they have been described before. Thus, the predicted rating now becomes:

$$\hat{r_{xi}} = \mu + b_x + b_i + p_x \cdot q_i$$

The objective function now becomes:

$$\min_{P,Q} \sum_{(i,x)\in R} (r_{xi} - p_x \cdot q_i)^2 + [\lambda_1 \sum_x ||p_x||^2 + \lambda_2 \sum_i ||q_i||^2 + \lambda_3 \sum_x ||b_x||^2 + \lambda_4 \sum_i ||b_i||^2]$$

We again apply the stochastic gradient descent algorithm and the new algorithm now becomes:

For each rating r_{xi}:

$\epsilon_i = r_{xi} - p_x \cdot q_i$

$q_i \leftarrow q_i + \mu_1(\epsilon_{xi} p_x - \lambda_2 q_i)$

$p_x \leftarrow p_x + \mu_2(\epsilon_{xi} q_i - \lambda_1 p_x)$

$b_i \leftarrow b_i + \mu_3(\epsilon_{xi} - \lambda_3 b_i)$

$b_x \leftarrow b_x + \mu_4(\epsilon_{xi} q - \lambda_4 b_x)$

5.3 Movie ratings prediction experimentation with LF

K-value Tuning

After implementing the algorithm we started testing it by running it with the 80% training data and calculating the RMSE for the 20% test data for the 100K dataset. There were several parameters that we had to tune to get the best results. Firstly we had to decide the number of factors that we want our model to have. We experimented with several values of k ranging from 1 to 200 and analyzed the results. Below you can see the results of the RMSE for the training and test data for different k values.

	K = 1	K = 2	K = 3	K = 4
Train RMSE	0.91429354	0.89435714	0.87895216	0.87311604
Test RMSE	0.94325129	0.94502097	0.94707306	0.93930841

Table 8: RMSE values for different k (dimensions) values

	K = 10	K = 40	K = 100	K = 200
Train RMSE	0.78133448	0.53834619	0.31232629	0.16297247
Test RMSE	0.93930841	1.04847543	1.25897100	1.88103736

Table 9: RMSE values for different k (dimensions) values

What we notice is that as k slowly increases the RMSE values on both the training and test data do not have a significant change however this changes as k gets very large. We notice that the larger the k value the smaller is the RMSE in the Training set and the larger is the RMSE in the Test set. This clearly shows the overfitting problem. As k gets very large the model overfits the training data and does not generalize for unseen data. We notice that for k=200 the error rate on the Test data is extremely small and on the Training data is extremely large. In figure 5 you can see a plot describing this behavior.

Learning Rate Tuning

Next we started experimenting with the learning rate value. We ran the algorithm with different learning rate values and observed that the best result was taken for the learning rate $\mu = 1e - 3$. Below you can check the Training and Test RMSE results for different learning rates.

	$\mu = 1e-5$	$\mu = 1e-4$	$\mu = 1e-3$	$\mu = 1e-2$
Train RMSE	0.94755633	0.86319259	0.79679054	0.81032666
Test RMSE	0.92819226	0.93033857	0.95335165	0.99245720

Table 10: RMSE values for different learning rate values

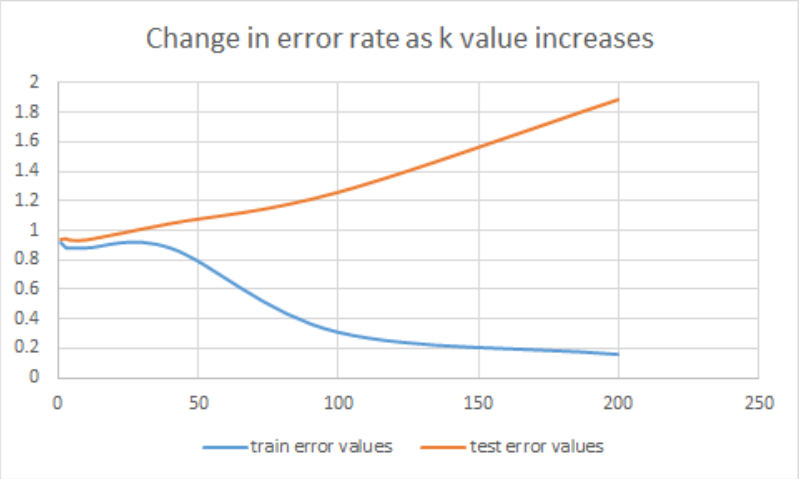

Figure 5: Change in error rate as k value increases.

Cross Validation

We did a 5-fold Cross validation of our algorithm. We repeated the experiment on 5 disjoint test sets and observed the values obtained. We averaged the results to gain an insight on the average RMSE for the whole dataset. We did this experiment on both the algorithms with and without bias. We used these values for the parameters: $k = 4$, $\mu = 0.005$, $\lambda_1 = 0.05$ and $\lambda_2 = 0.05$. Below you can see the results of this experimentation. We observe that the average RMSE on the Training set is 0.8365258 and the average RMSE on the Test data is 0.9524254. We did the same for the latent factor with baseline algorithm. In Table 12 we show the results. In this case the error average rate for the Training data is 0.7800806 and on the Test data 0.9397096 The results that we received were as we expected. The error rate for the Training set on both cases was relatively low. The error rate for the Test set which is unseen data is higher but it is significantly low proving the accuracy of our algorithm. The error rate on the Latent Factor with Bias experimentation was slightly lower than the error rate in the algorithm without bias which confirms the value of adding the bias factor to the prediction algorithm.

	Set 1	Set 2	Set 3	Set 4	Set 5
Train RMSE	0.817069	0.815156	0.807614	0.814456	0.805664
Test RMSE	0.949901	0.942070	0.937931	0.948221	0.944134

Table 11: RMSE values on 5 different subsets for Training and Test data - Without baseline

	Set 1	Set 2	Set 3	Set 4	Set 5
Train RMSE	0.777280	0.787899	0.779198	0.781458	0.774568
Test RMSE	0.959312	0.935591	0.933445	0.935657	0.934543

Table 12: RMSE values on 5 subsets for Training and Test data - With baseline

Running on bigger datasets

We further ran our algorithm on a dataset of 1 million ratings with 3883 movies and 6040 users. The parameters that we used are $k = 4$, $\mu = 5e - 3$, $\lambda_1 = 0.05$ and $\lambda_2 = 0.05$. We did 5-fold cross validation on this dataset as well. As previously described in the item-based section we used two different splittings into training and test data for this experimentation as well. Below we show each respectively. The results on the Training data, as expected are not different on either experiment. However, the RMSE on the Test data for the ordered splitting is higher than the RMSE on the Test data for the random splitting. This is explained by the fact that in the ordered splitting, the majority of the users which are on the Test (unseen) data is not present in the Training data, so the algorithm is not trained with any data regarding these users. Thus, it behaves as expected, not being very accurate with the predictions and showing a high RMSE. However, when trained on the random Training data, the RMSE for the Test data is dropped to around 0.8705, showing high accuracy in the predictions for unseen data. We notice the trend that as the number of ratings and data we provide to the algorithm increases, the accuracy on the predictions also increases.

	Set 1	Set 2	Set 3	Set 4	Set 5
Train RMSE	0.833958	0.834867	0.834732	0.836228	0.832151
Test RMSE	1.105338	1.106501	1.117193	1.106237	1.112317

Table 13: RMSE values on the 1M dataset - Ordered subsets

	Set 1	Set 2	Set 3	Set 4	Set 5
Train RMSE	0.830996	0.829556	0.830249	0.829894	0.80864
Test RMSE	0.867854	0.870592	0.870447	0.870762	0.874425

Table 14: RMSE values on the 1M dataset - Random subsets

We performed the same experiment on the dataset of 10 million ratings and obtained the results shown in table 15 and 16. The same trends explained above are proved by the results obtained from this experiment as well.

	Set 1
Train RMSE	0.789789
Test RMSE	1.09095

Table 15: RMSE values on the 10M dataset - Ordered subsets

	Set 1
Train RMSE	0.785494
Test RMSE	0.845421

Table 16: RMSE values on the 10M dataset - Random subsets

Making recommendations

In order to observe the accuracy of our predictions we run an experiment on the first user in the dataset. Using the above algorithm we make all the predictions of the ratings for all items for the first user. Next, we order these items

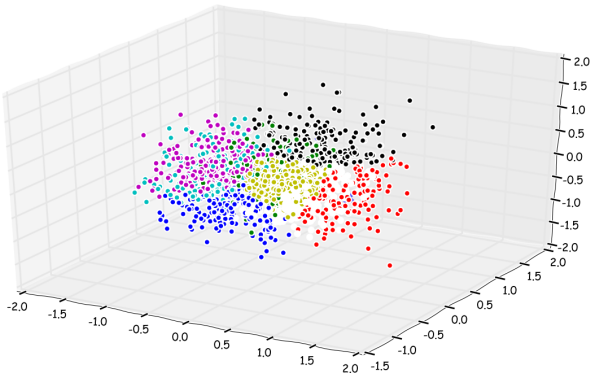

Figure 6: Clustering when k = 8.

in terms of the ratings in descending order and visualize the top 30 highest predicted rated items. We notice a similarity between the items recommended. And indeed, when searching in IMDB for some of these movies, we get as recommendations movies also present in the prediction set.

Dimensionality reduction and clustering

In order to understand better the Latent Factor algorithm we decide to visualize the dimensionality of the items vector. In order to visualize the k dimensions we firstly apply Principal Component Analysis to reduce the dimensionality of the k-dimensional items vector to 3 dimensions. Next we apply the kmeans clustering algorithm to cluster it into k dimensions. In order to visualize this clustering we plot it in 3D using using a different color for each cluster. In figure 6 we show this clustering for an 8-dimensional items vector. Lastly, we displayed the 20 closest items to the center of the first cloud and observed a relation in the genres of these items.

6 Conclusion

After experimenting with all above mentioned collaborative filtering algorithms on different sized datasets we reached several conclusions. Firstly, we observed that the as the size of the dataset used increased the RMSE value decreased. For the k-nearest neighbor collaborative filtering algorithms, we observed that the larger the value of k is, the better the RMSE resulted, and we assume that this stems from the sparsity of the ratings matrices. Additionally, in the Latent Factor algorithm we observed that a higher value of the k dimensions would lead to lower RMSE on Training data and higher RMSE on Test data. This shows the overfitting problem where a large number of features result in less generalizing for unseen data.

Team Work

Leonard Dervishi, Syed Sarjeel Yusuf - Implementation and analysis of User-user algorithm and Slope One algorithm
Figali Taho - Implementation and analysis of Item - item CF algorithm
Anisa Llaveshi - Implementation and analysis of Latent Factor Algorithm

References

[1] MovieLens, *GroupLens*. http://grouplens.org/datasets/movielens/

[2] GroupLens, *GroupLens*. http://files.grouplens.org/papers/FnT

[3] Zhouxiao Bao, Haiying Xia Movie Rating Estimation and Recommendation

[4] Yashodhan Karandikar Movie Rating Prediction using the MovieLens dataset

[5] Data Piques, Explicit Matrix Factorization: ALS, SGD, and All That Jazz. http://blog.ethanrosenthal.com/2016/01/09/explicit-matrix-factorization-sgd-als/.

[6] Ron Zacharski. Guide to data mining. http://guidetodatamining.com/

[7] Sarwar, Badrul, et al. "Item-based collaborative filtering recommendation algorithms." Proceedings of the 10th international conference on World Wide Web. ACM.

[8] A. Rajaraman and J. D. Ullman, Mining of Massive Datasets, Cambridge University Press, 2011