EE 219 Project III - Report

Muralidharan, Vignesh UID: 904729596
Muthappan, Chidambaram UID: 704774938
Jeyakumar, Jeya Vikranth UID: 404749568

Abstract

Recommendation systems are like a family of methods that enable filtering through large observation and information space. This can be used to provide recommendations in the information space that user does not have any observation, where the information space is all of the available items that user could choose or select and an observation space is what user experienced or observed so far.

Collaborative Filtering is a subset of algorithms that use other users and items along with their ratings and target user history to recommend an item that target user does not have ratings for. The fundamental assumption behind this approach is that other users preference over the items could be used recommending an item to the user who neither saw the item nor purchased it before.

The purpose of this project is to build a recommendation system with "Alternating Least Squares", which is basically a popular method for collaborative filtering.

1 Question 1

The MovieLens dataset consists of a total of 100k ratings. We created a matrix R from this dataset in such a way that the rows corresponded to the users (user-id) and columns corresponded to the movies (movie-id). From the dataset, we identified that the number of users were 943 and the number of movies were 1682 respectively. Therefore, we made sure that the matrix R we created consisted of 943 rows and 1682 columns.

Each entry into the R matrix signified the rating that a particular user has given to a particular movie. This effectively means that R[i][j] would hold the rating given by the ith user to the jth movie. If user i had not rated movie j, the entry at R[i][j] was marked to be NaN (as stated in wnmfrule Toolbox's function definition). It was also noticed that, since not all users had rated all movies, there were a significantly large number of missing data points found in the matrix.

After this step, we created a weight matrix 'W' with dimensions 943 x 1682, i.e. same as matrix R. The entries in the weight matrix depended on whether or not a user had rated a certain movie. In other words, W[i][j] was marked

to be 1 if user i had rated movie j, and NaN otherwise (as stated in wnmfrule Toolbox's function definition).

The ultimate aim was now to find such matrices U and V by the method of matrix factorization, that the squared error is minimized. In order to achieve this, the function wnmfrule from the Matrix Factorization Toolbox available in Matlab was used. We passed the matrix R along with the dimension value k as input parameters to the function. After passing, as an output, the function factorized the matrix R into two matrices U and V, and also returned a residual error.

The error of an observed value is the deviation of the observed value from the (unobservable) true value of a quantity of interest (for example, a population mean), and the residual of an observed value is the difference between the observed value and the estimated value of the quantity of interest.

As per given formula, The least squared error can be calculated as:

$$min \sum_{i=1}^{m} \sum_{j=1}^{n} w_{ij}(r_{ij} - (UV)_{ij})^2 \tag{1}$$

We repeated the factorization for 3 different values of k :[10,50,100]. The following table represents the results (residual error, squared error) for different dimensions and iterations.

Table 1: Recommendation System

Dimension k	Residual Error	Least Squared Error	Iterations
10	232.9665	5.4273 e+04	1000
50	139.6195	1.9494 e+04	1000
100	79.4198	0.6308 e+04	1000

Also, from Table 1, it can be seen that:

- Residual error decreases with increase in the dimension 'k'. From the 'wnmfrule.m' Matlab file which was used, we observed that 'k' refers to the number of clusters. We have 943 users and we're trying to map all of them in 10 clusters. In that case, the probability of users with different tastes grouped into the same cluster is high. So, the residual error is higher for k=10 than k=100.
- Lest Squared error decreases with increase in the dimension 'k'. From the 'wnmfrule.m' Matlab file which was used, we observed that 'k' refers to the number of clusters. We have 943 users and we're trying to map all of them in 10 clusters. In that case, the probability of users with different tastes grouped into the same cluster is high. So, the least squared error is higher for k=10 than k=100.

Thus, the optimal least squared error was obtained at k = 100.

2 Question 2

From the scikit-learn documentation:
Cross-validation is a model validation technique used for assessing how the results of a statistical analysis will generalize to an independent data set. The goal of cross validation is to define a dataset to "test" the model in the training phase (i.e., the validation dataset), in order to limit problems like over fitting. One of the main reasons for using cross-validation instead of using the conventional validation is that there is not enough data available to partition it into separate training and test sets without losing significant modeling or testing capability. In these cases, a fair way to properly estimate model prediction performance is to use cross-validation as a powerful general technique.

But in the given dataset one cannot remove rows or columns of the matrix entirely because that would jeopardize the idea behind this method. So to overcome this problem it is better to we assign an index 1 to N to the N known data points. Create a random ordering of the numbers 1 to N and split this list of numbers into 10 part. The R matrix then can be generated on the 90% of chosen data points. This training dataset is then passed through the matrix factorization method for k equal to 10,50 and 100.

The absolute error over the testing data is given by

$$|R_{predicted} - R_{actual}| \tag{2}$$

The average absolute error for one fold is then calculated by

$$(\sum_{i=1}^{size(testdata)} |R_{i,predicted} - R_{i,actual}|)/size(testdata) \tag{3}$$

These metrics can then be used to calculate the average error for each system (k=10,50 and 100). It is important to note that Weighted Non-negative Matrix Factorization (wmnf rule) is used to generate the recommendation systems.

Table 2: Absolute error for Systems

Fold	K = 10	K = 50	K = 100
1	8784.20081	16549.73596	11132.5447
2	5717.21208	3156.235465	1623.33912
3	5703.625662	3154.675314	1638.913005
4	5684.139445	3185.718869	1647.254909
5	5718.992315	3225.403862	1653.052411
6	5692.245228	3160.012693	1638.201124
7	5696.725669	3198.855966	1635.197678
8	5768.138097	3198.134388	1655.940836
9	5783.763634	3217.654882	1667.058757
10	5717.279667	3212.619804	1648.638122

Table 3: Minimum and Maximum Average Errors of the Systems

	K = 10	K = 50	K = 100
Average	0.602	0.452	0.259
Minimum	0.878420081	1.654973596	1.11325447
Maximum	0.568413944	0.315467531	0.162333912

Table 4: Average error for Systems

Fold	K = 10	K = 50	K = 100
1	0.878420081	1.654973596	1.11325447
2	0.571721208	0.315623547	0.162333912
3	0.570362566	0.315467531	0.163891301
4	0.568413944	0.318571887	0.164725491
5	0.571899232	0.322540386	0.165305241
6	0.569224523	0.316001269	0.163820112
7	0.569672567	0.319885597	0.163519768
8	0.57681381	0.319813439	0.165594084
9	0.578376363	0.321765488	0.166705876
10	0.571727967	0.32126198	0.164863812

Therefore, table 3 also shows that Average error for K = 100 is the least among the three systems.

3 Question 3

The task in this particular part is to plot the ROC curve for a different values of threshold. It is mentioned in the question that the user does not like the movie if he has rated the movie with a 3 or lower on a scale of 5. If otherwise, the user likes the movie.

To get a better idea of the whole system, we tried to analyze the system by incrementing threshold over a scale of 0.5 with 1 as our beginning point and 5 as our end point. There are several advantages in analyzing the threshold on a scale of 0.5 than 1. Firstly, it will be easier to see the the number of entries in our system where the user would like the movie. Based on precision and accuracy values obtained, it will become easier to see which threshold value can be fixed.

Before we start looking at the results for these different algorithms let us first understand the various Model Evaluation Metrics based on which the algorithms are analyzed.

Confusion matrix: Table that describes the performance of a classification model Every observation in the testing set is represented in exactly one box It's a 2x2 matrix because there are 2 response classes Confusion matrix gives you a more complete picture of how your classifier is performing Also allows you to compute various classification metrics, and these metrics can guide your model

Table 5: Residual of the Systems

Fold	K = 10	K = 50	K = 100
1	218.8272113	123.4851694	64.21706034
2	233.6261406	138.9120227	78.63984788
3	233.2530323	139.8360948	78.66098587
4	233.2777949	138.9724055	78.67151269
5	233.4577392	139.425731	79.0151585
6	233.1492367	139.3783724	78.62470676
7	233.1074202	139.7462097	78.49746966
8	232.6623432	139.0784115	78.7938031
9	233.2271585	139.5713119	78.71548162
10	234.2078079	139.4788045	79.37268949

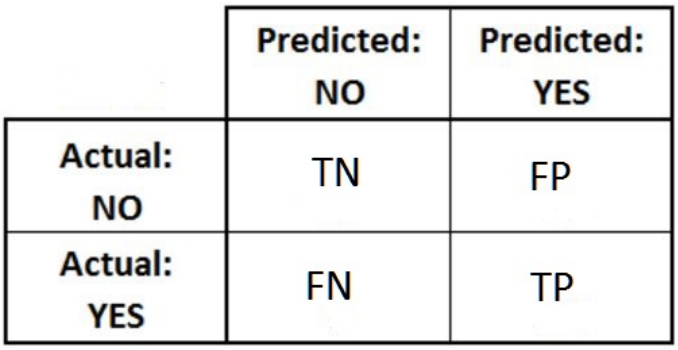

Figure 1: Confusion Matrix

selection
Basic terminologies:

- True Positives (TP) - Correctly predicted that they belong to Class 0
- True Negatives (TN) - Correctly predicted that they belong to Class 1
- False Positives (FP) - Incorrectly predicted that they belong to Class 0
- False Negatives (FN) - Incorrectly predicted that they belong to Class 1

Classification accuracy: percentage of correct predictions
Classification accuracy is the easiest classification metric to understand but, it does not tell us the underlying distribution of response values. It also does not tell us what "types" of errors our classifier is making.

$$Accuracy = (TP + TN)/(TP + TN + FP + FN) \tag{4}$$

Recall/True Positive Rate: Tells us how often the prediction is correct When the actual value is positive.Also known as "True Positive Rate" or "Sensitivity"

$$Recall = TP/(TP + FN) \tag{5}$$

Precision: Tells us how often the prediction is correct when a positive value is predicted.

$$Precision = TP/(TP + FP) \tag{6}$$

False Positive Rate: Tells us how often the prediction is incorrect when the actual value is negative.

$$FPR = FP/(TN + FP) \tag{7}$$

ROC Curves: Tells us how sensitivity and specificity are affected by various thresholds, without actually changing the threshold.

Area Under the Curve (AUC): AUC is the percentage of the ROC plot that is underneath the curve.AUC is useful as a single number summary of classifier performance.

We first tried to find the precision, recall and accuracy for different values of threshold and for k = [10, 50, 100].

Table 6: Precision, Recall and Accuracy for K = 10

Threshold	Precision(%)	Recall(%)	Accuracy(%)
1	56.0436	99.8381	56.38
1.5	56.6353	99.4963	57.37
2	57.8237	98.6508	59.25
2.5	60.4884	95.8086	62.88
3	65.8107	89.4405	68.30
3.5	72.4477	71.6136	69.08
4	80.3152	44.9182	63.26
4.5	83.1652	19.2840	52.96
5	78.2609	5.1808	46.49

From the confusion matrix for K = 10 (Figure 2), the following values can be obtained:

The most optimum values for K = 10 are obtained for Threshold = 3.

Precision = 65.8107 %

	Predicted: NO	Predicted: YES
Actual: NO	1858	2583
Actual: YES	587	4972

Figure 2: Confusion Matrix for K = 10

Recall = 89.4405 %

Accuracy = 68.30 %

Total Number of predicted likes = 7555

Total Number of actual likes = 5559

Table 7: Precision, Recall and Accuracy for K = 50

Threshold	Precision(%)	Recall(%)	Accuracy(%)
1	56.2703	99.6042	56.75
1.5	57.2083	98.7947	58.25
2	58.9174	96.3303	60.62
2.5	61.4099	91.2035	63.25
3	64.9407	80.7699	65.07
3.5	69.1824	65.2995	64.54
4	73.1763	45.8356	60.55
4.5	75.4220	27.3251	54.65
5	74.1245	13.7075	49.37

From the confusion matrix for K = 50 (Figure 3), the following values can be obtained:

Precision = 64.9407 %

Recall = 80.7699 %

Accuracy = 65.07 %

Total Number of predicted likes = 6914

	Predicted: NO	Predicted: YES
Actual: NO	2017	2424
Actual: YES	1069	4490

Figure 3: Confusion Matrix for K = 50

Table 8: Precision, Recall and Accuracy for K = 100

Threshold	Precision(%)	Recall(%)	Accuracy(%)
1	56.1316	99.7122	56.52
1.5	57.0313	98.4889	57.91
2	58.4532	95.0351	59.69
2.5	61.0454	88.2353	62.16
3	63.9225	75.9849	62.81
3.5	67.5599	59.8669	61.71
4	69.9609	41.8960	57.70
4.5	70.5479	25.9399	52.81
5	70.4348	14.5710	49.11

Total Number of actual likes = 5559

From the confusion matrix for K = 100 (Figure 4), the following values can be obtained:

Precision = 63.9225 %

Recall = 75.9849 %

Accuracy = 62.81 %

Total Number of predicted likes = 6608

Total Number of actual likes = 5559

From the above analysis, it can be found that:

- Most optimum value is obtained for **K = 10** and **Threshold = 3**

	Predicted: NO	Predicted: YES
Actual: NO	2057	2384
Actual: YES	1335	4224

Figure 4: Confusion Matrix for K = 100

- **Entries predicted where user would like the movie = 7555**
- **Precision = 65.8107 %**
- **Test data entries where user did actually like the movie = 5559**
- **Recall = 89.4405 %**

ROC curve is plotted in Figure 5. Prevision vs Recall is plotted in Figure 6. Area under the ROC curve for K=10 is **0.6805**

4 Question 4

In this part, we changed the weight matrix created in part 1 by assigning the actual ratings to weight matrix, while the values in R matrix are changed to 0 and 1 i.e. for any valid entry the value of R matrix is changed to 1 and for any unknown entry the value is changed to NaN.

So, for the new weight and R matrix we calculated the least squared error and we found it to be lower than the one we got in part 1. Now, as given in the problem, in order to avoid singular solutions, we modified the cost function by adding a regularization term as per:

$$min \sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}(r_{ij} - (UV)_{ij})^2 + \lambda(\sum_{i=1}^{m}\sum_{j=1}^{k} u_{ij}^2 + \sum_{i=1}^{k}\sum_{j=1}^{n} v_{ij}^2) \tag{8}$$

In order to calculate these values, we pass the weight and R matrices along with the dimension value k as input. As an output, the function factorized matrix R into two matrices U and V, and also returned a residual error.

We use regularization because this term forces all the entries to shrink, but along with this there is a countering term that extracts the predicted value of

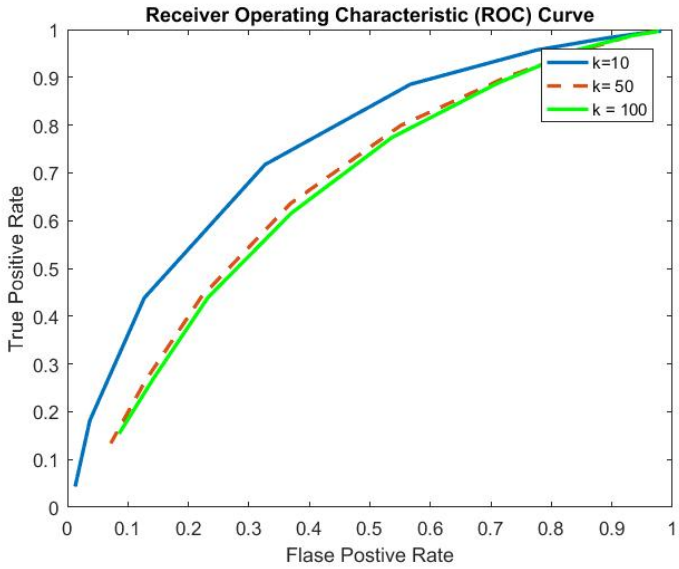

Figure 5: ROC Curve for K = 10, 50, 100

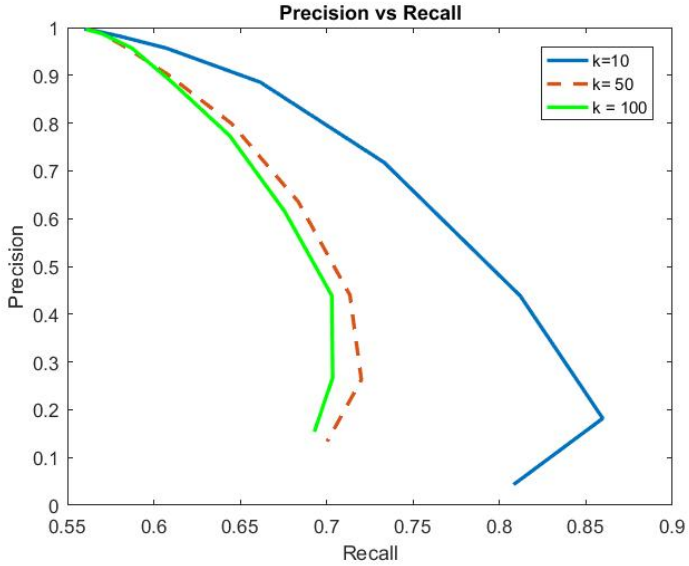

Figure 6: Precision vs Recall for K = 10, 50, 100

$$U = U.*(((W.*R)*V')./(\lambda*U + (W.*(U*V))*V'))$$
$$V = V.*((U'*(W.*R))./(\lambda*V + U'*(W.*(U*V))))$$

Figure 7: Formula

the entries that have an actual rating. This term is more significant for those entries with higher ratings. Due to this, the predicted matrix will have higher values for the entries that are more likely to be favorable. The above equation was solved using Alternating Least Square Function where the matrix U and V are iteratively updated such that the residual error is minimized. The values of U and V are obtained using wnmfrule function. This function calculates the values of U and V as given in Figure 7.

Table 9: For normal R & W

k	Least Squared Error
10	5.4273 e+04
50	1.9494 e+04
100	0.6308 e+04

Table 10: For swapped R & W

k	Least Squared Error
10	0.504654
50	2.875609
100	6.520043

Observation:

- We could see that the regularization helps make the precision vs recall curve smoother. So is the case for swapped R and W.
- **Figures 8, 9 and 10** show the ROC curves for different values of λ

5 Question 5

In this part, we created the matrix R in such a way that the entries in R were binary in nature, i.e., R[i][j] was marked to be 1 if user i had rated the movie j, and NaN otherwise. On the other hand, the weight matrix comprised of the actual ratings that the users had given. We then performed a 10 fold cross validation, as in part 2, and formed a new matrix called 'predicted' that kept a track of the predicted ratings corresponding to the known data points.

Now, predicted ratings for each user in the predicted matrix were sorted in a descending order and their corresponding movie ids were stored separately.

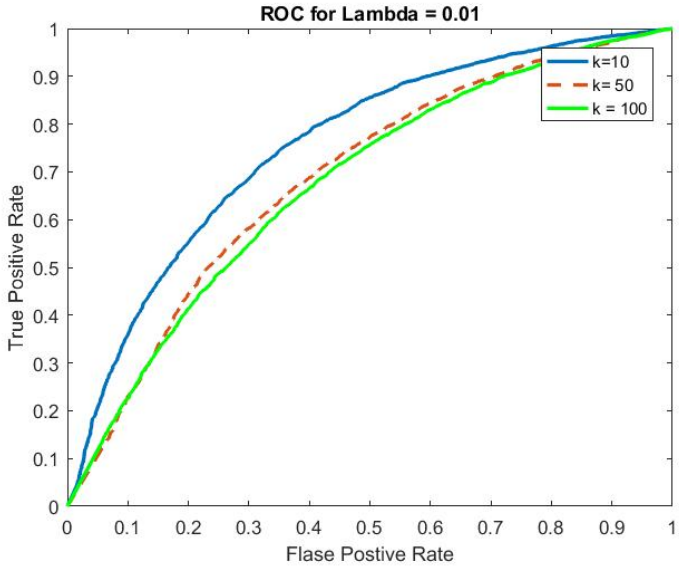

Figure 8: ROC for $\lambda = 0.01$ [K = 10, 50, 100]

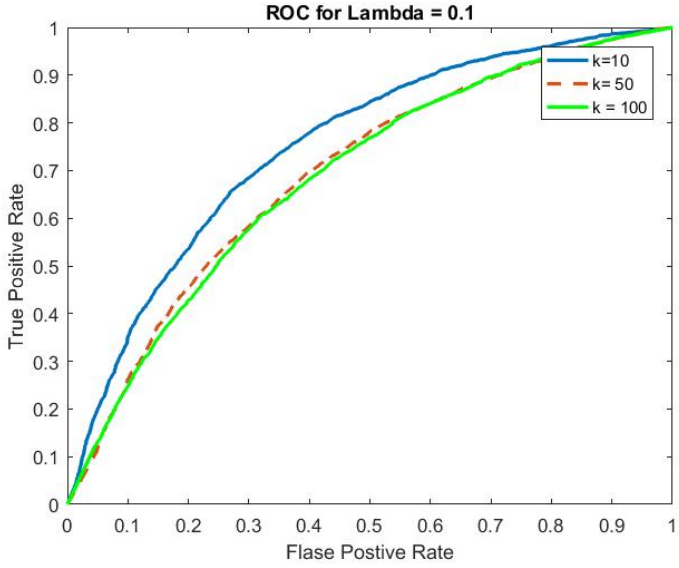

Figure 9: ROC for $\lambda = 0.1$ [K = 10, 50, 100]

Table 11: For Swapped R & W with Regularization

	K = 10	K = 50	K = 100
$\lambda = 0.01$	44.696305	919.881640	11327.615044
$\lambda = 0.1$	32.894697	337.456335	3381.9116152
$\lambda = 1$	102.110588	342.473630	2822.296866

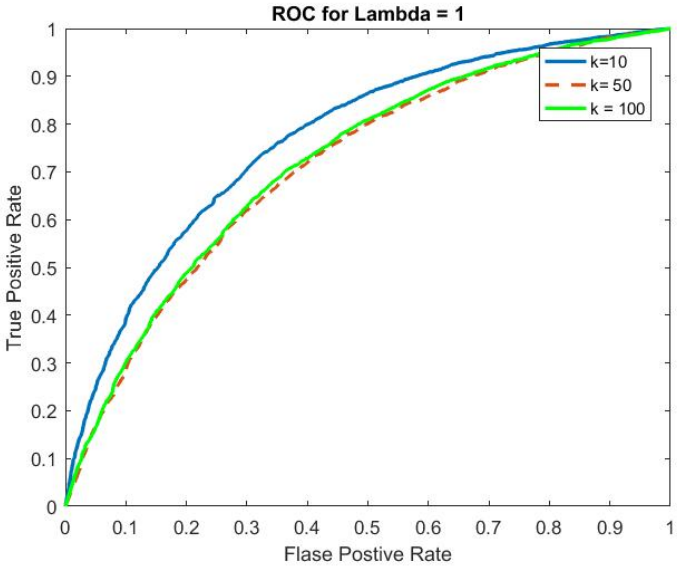

Figure 10: ROC for $\lambda = 1$ [K = 10, 50, 100]

This allowed us to select the top L movies in a much easier manner. We started with L=1 and kept on increasing it till it reached 20. For each value of L, we created a 'top movies' matrix that stored the top 'L' movies for each user.

In order to measure how well our algorithm was doing, we calculated the algorithm's hit-rate and false-alarm rate for each L by using the following formulae:

$$HitRate = \frac{TruePositive}{TruePositive + FalseNegative} \tag{9}$$

$$FalseAlarmRate = \frac{FalsePositive}{FalsePositive + TrueNegative} \tag{10}$$

where, true positive = no. of times both actual and predicted ratings are above the threshold true negative = no. of times both actual and predicted ratings are below the threshold false positive = no. of times the actual rating is above the threshold while the predicted rating is not false negative = no. of times the actual rating is below the threshold while the predicted rating is not

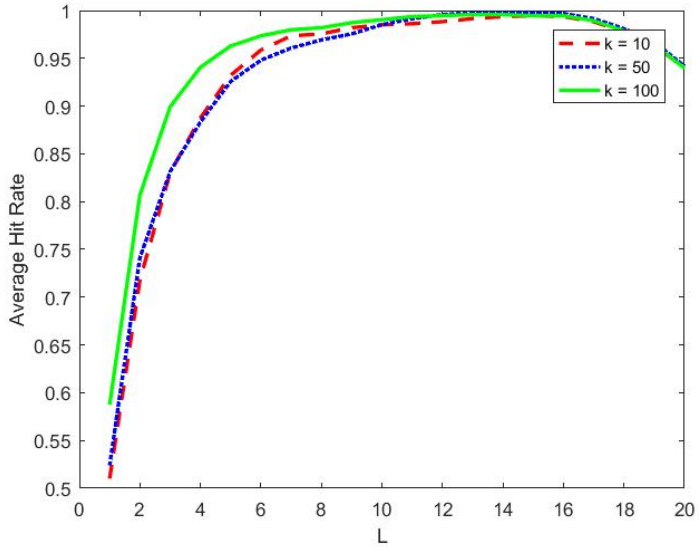

Figure 11: Average Hit Rate vs L for [K = 10, 50, 100]

It should be noted that the actual ratings were compared against a ground threshold of 3 wherein a rating greater than or equal to 3 meant that the user liked the movie and less than 3 meant he disliked the movie. The predicted ratings were, on the other hand, compared against a normalized threshold that was set to 0.4 based on previously obtained observations. When the value of L hit 5, we calculated the average precision of the algorithm based on the following formula:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{11}$$

The entire process was repeated for k = 10, 50 and 100, and the following results were obtained:

Table 12: Average Precision for L = 5

Dimension k	Average Precision
10	0.5307
50	0.5435
100	0.5632

As we can notice, the average precision increases with an increase in the value of k and when we set the dimension k = 100, we achieved the highest precision of 0.5632.

As shown in Figure 11, we can observe from the above graph that as the value of L increases, the Hit Rate tends to approach 1, i.e. an optimal situation. This is because as L increases, the scope of the movies that are to be suggested

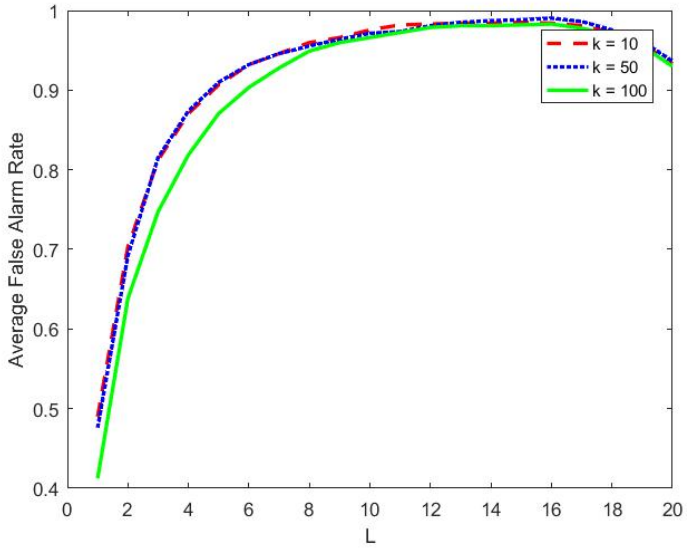

Figure 12: Average False Alarm Rate vs L for [K = 10, 50, 100]

to the user increases. If the value of L, i.e. the number of movies to be suggested to the user surpasses the number of movies that are actually liked by the user, it results into a hit rate value of 1 for that particular condition.

As show in Figure 12, we can observe that, similar to the case of Hit Rate vs L, as L increases, the False Alarm Rate also tends to reach 1. Again, if the value of L, i.e. the number of movies to be suggested to the user, surpasses the number of movies that are actually disliked by the user, it results into a false alarm rate value of 1 for that particular condition.

As shown in Figure 13, it is observed that, since both, Hit Rate vs L and False Alarm Rate vs L approach 1 with an increase in the value of L, the Hit Rate vs False Alarm Rate curve is also an increasing one in such a manner that it approaches 1.

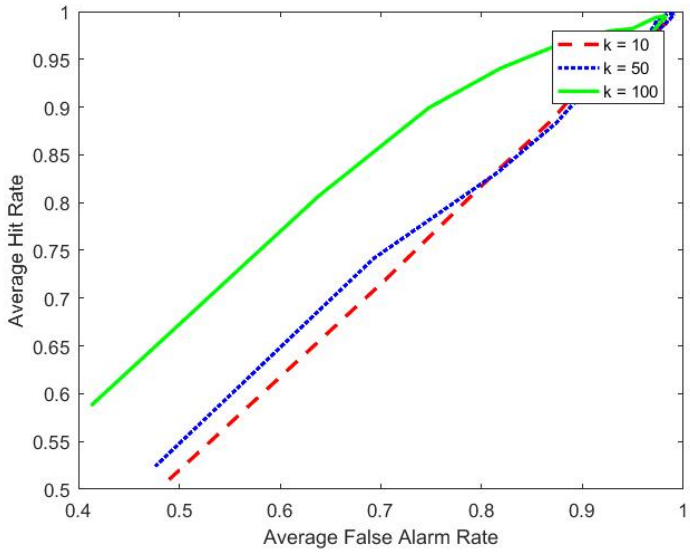

Figure 13: Average Hit Rate vs Average False Alarm Rate for [K = 10, 50, 100]