EE239 AS Project 3 Report
Collaborative Filtering
Winter 2016

Team members:
Rebecca Correia (204587944)
Salil Kanetkar (704557096)
Vedant Patil (104590942)

Part 1

The MovieLens dataset that we have been provided with for this project 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). We found from the dataset that the number of users and movies were 943 and 1682 respectively. Therefore, R consisted of 943 rows and 1682 columns.

Each entry into the R matrix signified the rating that each user had given to each movie. That 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. Since not all users had rated all movies, there were a significantly large number of missing data points found in the matrix.

Next, we created a weight matrix of 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.

The goal 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, we used the function wnmfrule from the Matrix Factorization Toolbox available in Matlab. We passed the matrix R along with the dimension value k as input parameters to the function. As an output, the function factorized the matrix R into two matrices U and V, and also returned a residual error. The least squared error could then be calculated as:

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

We repeated the factorization process for 3 different values of k: [10,50,100]. The observations that were made in each case are as follows:

Dimension k	Residual Error	Squared Error	Iterations
10	245.67	6.0355e+04	100
50	173.98	3.0269e+04	100
100	130.95	1.7147e+04	100

It can be noticed that as the value of k was increased, the least squared error kept decreasing. Thus, the optimal least squared error was obtained at k = 100.

Part 2

In this part, we first created a random ordering of indices ranging from 1 to 100000 using the 'crossvalind' function in MATLAB. This random ordering was then used to perform the 10 fold cross validation process.

During cross validation, we consider only a certain fraction of the data as the test data and the rest as the training data. In our case, for each fold, we used the randomly ordered indices to split the dataset in such a way that 1/10th of the data was considered to be the testing data and the remaining 9/10th as the training data.

Once the training matrices R and W were obtained, R was passed along with different values of k as input parameters to the wnmfrule function in order to obtain the matrix factorization. This was then compared against the testing data so as to yield the absolute error for each fold.

The average, maximum and minimum error for each of the 10 folds was observed to be the following for different values of k:

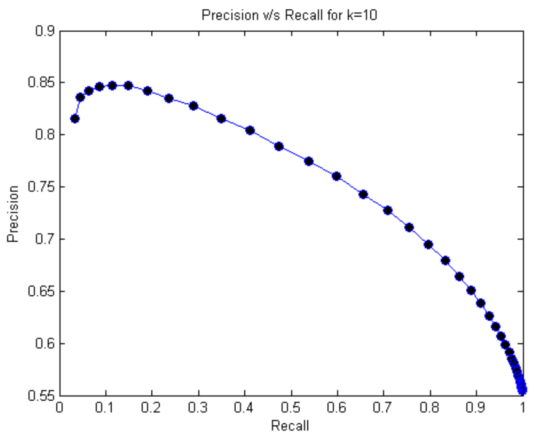
Dimension k	Average error	Maximum error	Minimum error
10	1.1943	4.8003	0.7861
50	0.9044	0.9658	0.8873
100	0.9028	0.9213	0.8938

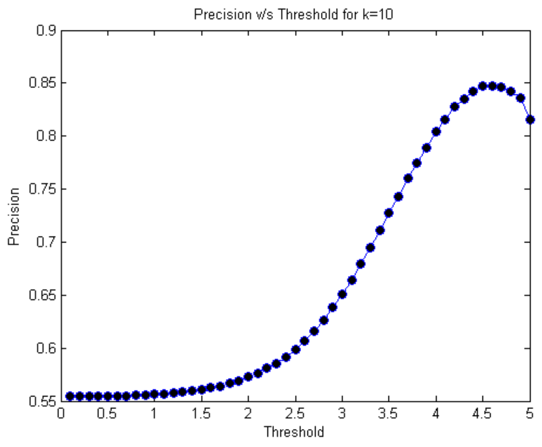
As we can see, the average and the maximum error went on decreasing with an increase in the value of k, however, on the other hand, the minimum error went on increasing as the value of k was increased from 10 to 100.

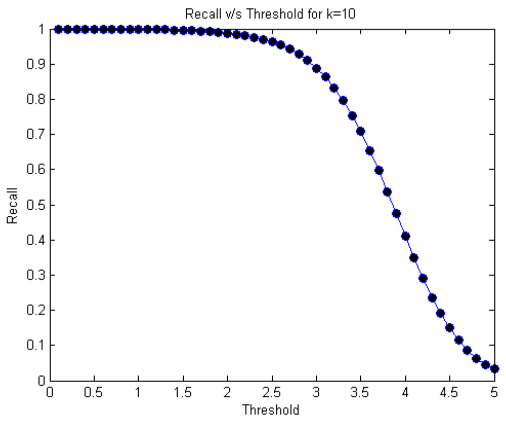
<u>Part 3</u>

For Part 3 our task was to find the precision and recall of our algorithm and plotting graphs for precision versus recall for different values of threshold for all the values of k=10, 50, 100. We consider 50 threshold values between 0.1 and 5 for the predicted values of ratings. For the actual ratings, we consider a fixed threshold of 3, and calculate precision and recall values. For each value of k below, the first graph is the precision v/s recall, second graph shows variations in the precision value as

For k=10

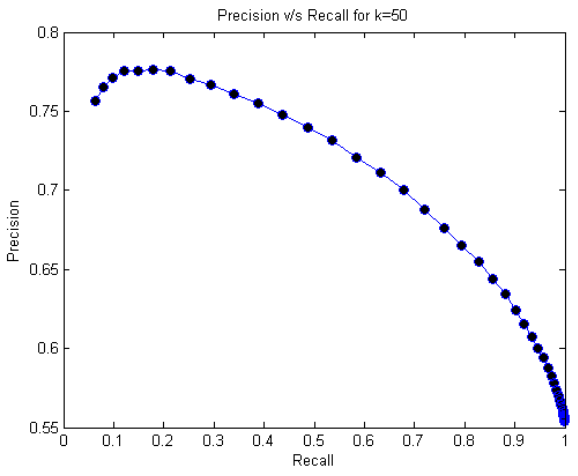


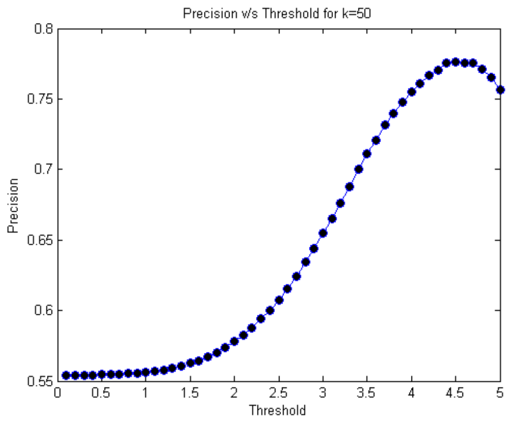


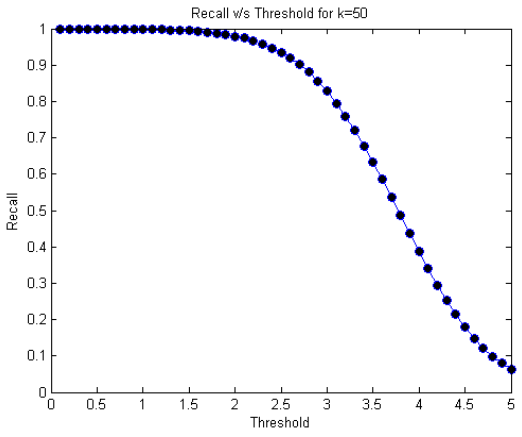


The area under the precision vs recall curve was: 0.7358

For k=50

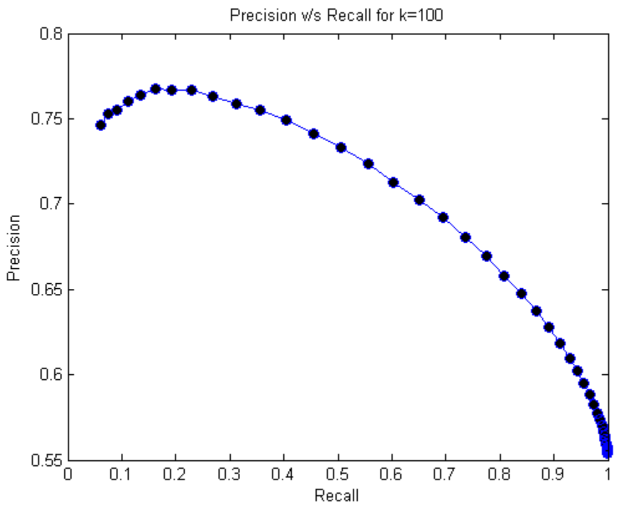


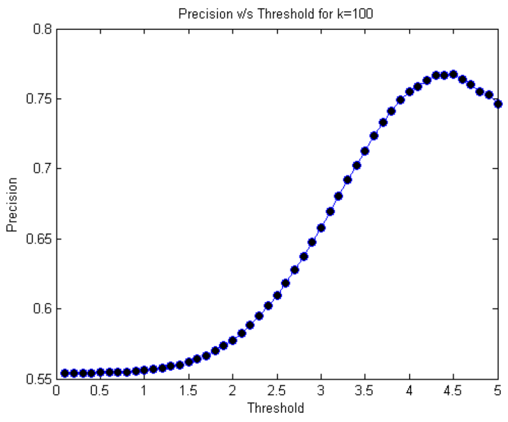


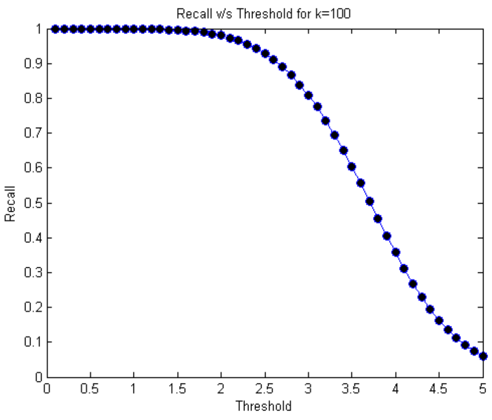


The area under the precision vs recall curve was: 0.6685

For k=100







The area under the precision vs recall curve was: 0.6660

Observations:

- The Precision versus Recall plot shows that Precision decreases as Recall increases, and with an increase in the value of k, the curve becomes further steep.
- As can be seen above, Precision increases with an increase in threshold value. The curve becomes steeper for an increase in the value of k.
- As can be seen above, Recall decreases with an increase in threshold value. The curve becomes steeper for an increase in the value of k.

<u>Part 4</u>

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:

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

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 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 reg_wnmfrule function. This function calculates the values of U and V as:

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

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.

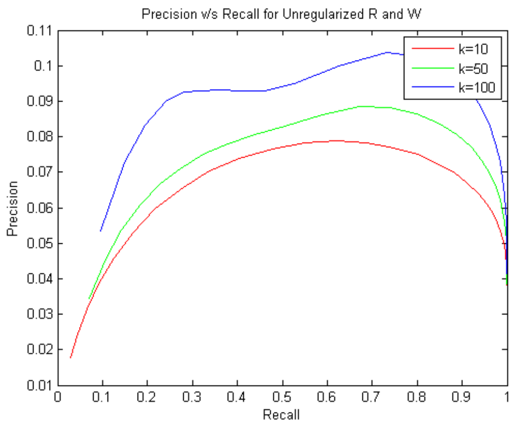
The test was conducted for 100 iterations and the results are as:

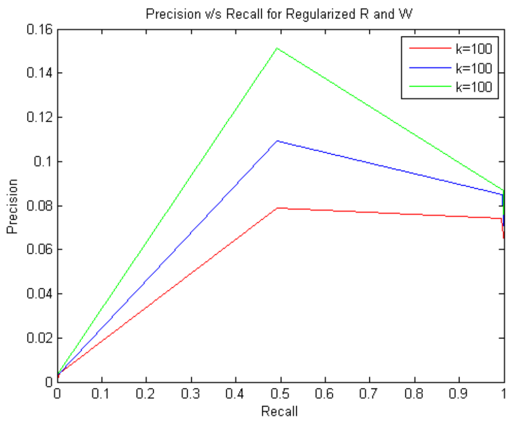
For normal R and W		For swapped R and W	
k	least squared error	k	least squared error
10	1.0e+05 *1.8377	10	248.2580
50	1.0e+05 *0.9089	50	556.8834
100	1.0e+05 *0.5181	100	441.7232

For regularized R and W			For regularized swapped R and W		
□	k	least squared error	□	k	least squared error
0.01	10	1.0e+04 * 6.0791	0.01	10	71.9654
	50	1.0e+04 * 6.0760		50	79.2524
	100	1.0e+04 * 6.1139		100	166.8976
0.1	10	1.0e+04 * 3.0302	0.1	10	180.9738
	50	1.0e+04 * 3.0642		50	175.2225
	100	1.0e+04 * 3.2037		100	242.0428
1	10	1.0e+04 * 1.7329	1	10	144.2015

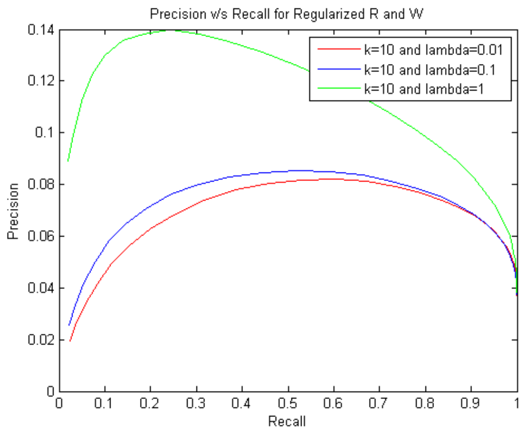
	50	1.0e+04 * 1.7546		50	164.3880
	100	1.0e+04 * 1.9227		100	202.7095

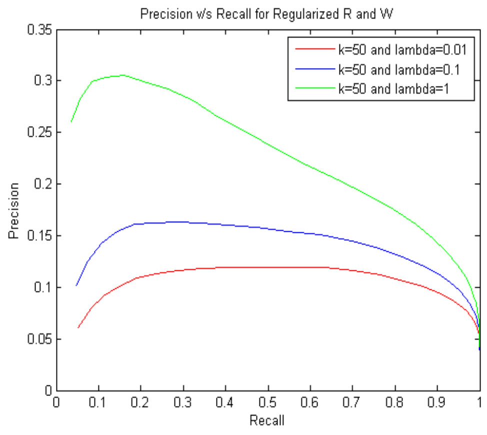
Let's see the precision v/s recall performance for unregularized R and W: The graph on the left is that for normal R and W, and the one on the right is where R and W are swapped and the prediction is made. These two graph would act as a benchmark for the regularization.

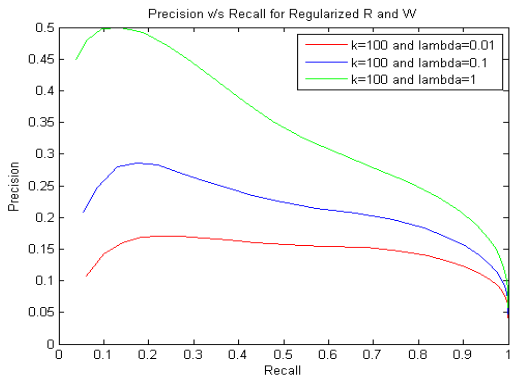




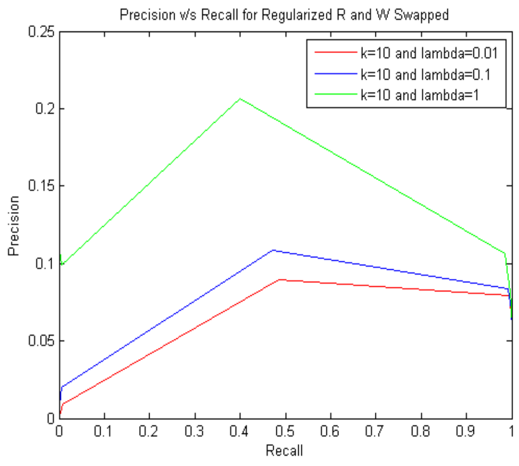
Let's see the precision v/s recall performance for regularized R and W:
The graphs below are for k=10,50,100 respectively.

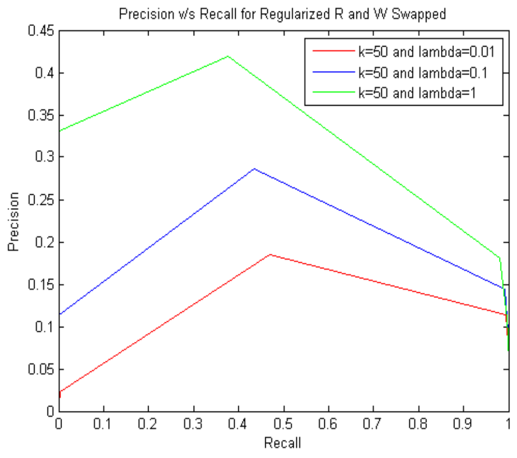


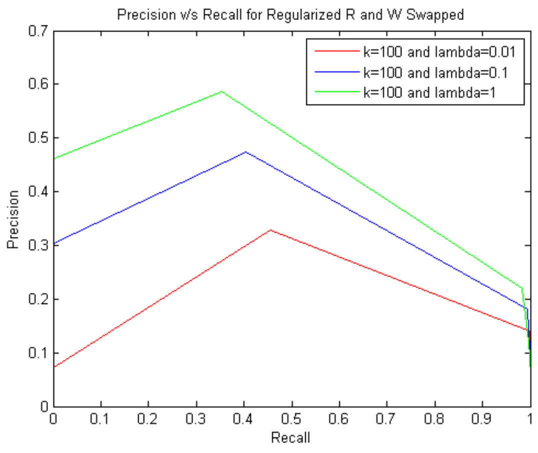




Now we also perform regularization by Swapping R and W. The graphs are as below:







Observations:
We see that the regularization helps make the precision vs recall curve smoother. So is the case for swapped R and W.

Part 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. 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:

$$\text{Hit rate} = \frac{true\ positive}{true\ positive + false\ negative}$$

$$\text{False alarm rate} = \frac{false\ positive}{false\ positive + true\ negative}$$

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

It should be noted that the actual ratings were compared against a ground threshold of 3 wherein a rating >= 3 meant that the user liked the movie and < 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:

$$\text{Precision} = \frac{true\ positive}{true\ positive + false\ positive}$$

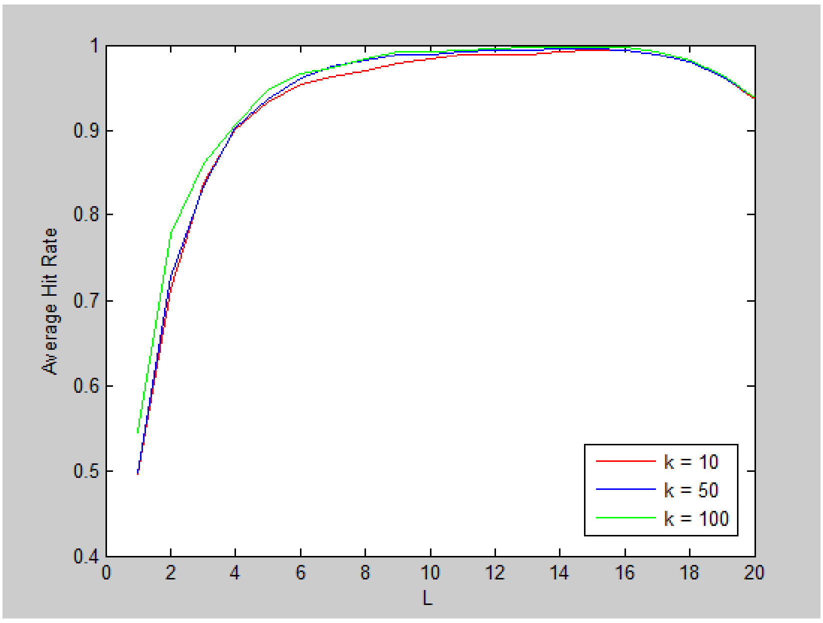
The entire process was repeated for k = 10, 50 and 100, and the following results were obtained:

Average precision for L = 5

Dimension k	Average precision
10	0.5306
50	0.5432
100	0.5631

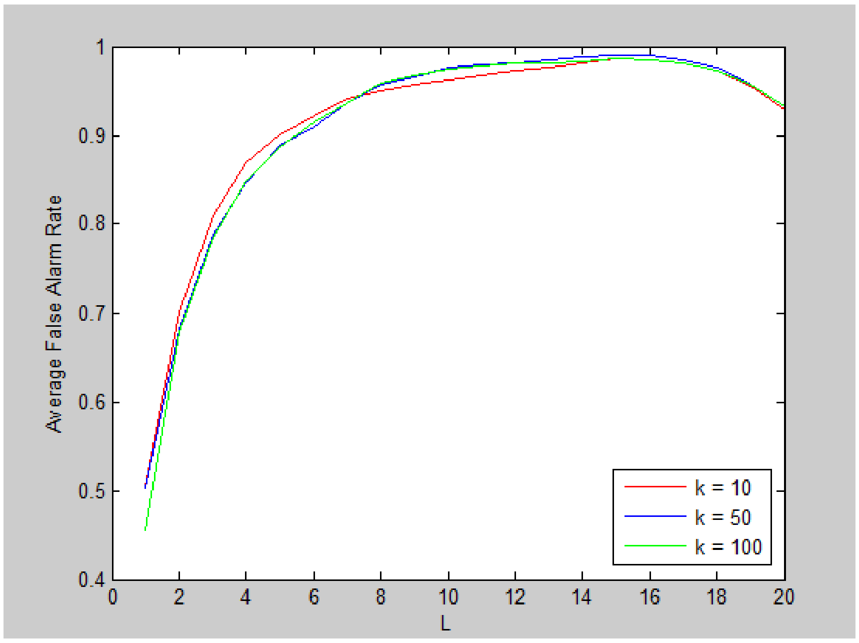
As we can notice, the average precision increases with an increase in the value of k and when we set the dimension k = 100, we achieved the highest precision of 0.5631.

Average Hit Rate vs L for k = [10, 50, 100]



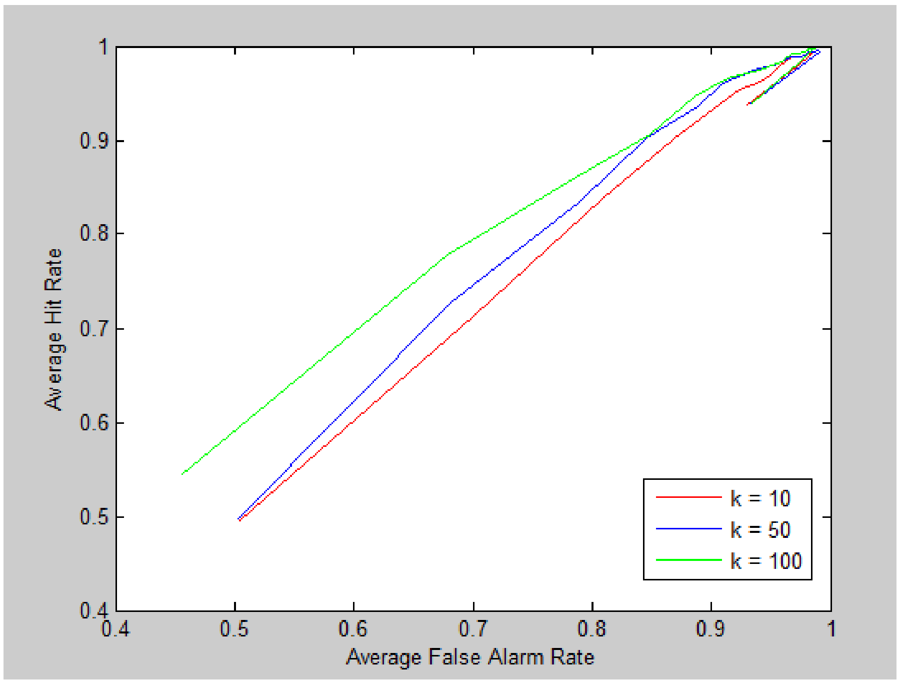
We can observe from the above graph that as the value of L increases, the Hit Rate tends to approach 1, i.e. an optimal situation. This is because as L increases, the scope of the movies that are to be suggested to the user increases. If the value of L, i.e. the number of movies to be suggested to the 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.

Average False Alarm Rate vs L for k = [10, 50, 100]



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.

Average Hit Rate vs Average False Alarm Rate for k = [10, 50, 100]



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.