EE219 Project 3
Collaborative Filtering

Winter 2017

Boyang Cai 304330123
Manni Chen 304145309
Zhuoqi Li 004855607

Problem I. Weighted Non-negative Matrix Factorization

Our objective of this project is to create a movie recommendation system that suggests movie to users based on their preference. This is done by recommending movies that were rated high by those who share similar tastes with the target user. In order to do this, we take in a original movie rating matrix that values in cell ij represent the rating user i gives to movie j. The rating values range from 1 to 5 if the user had rated the movie and 0 otherwise. Then we apply weighted non-negative factorization on the data matrix. The matrix factorization method generates two matrices R = U*V such that each cell of R is generated by dot product of latent vector describing user and a latent vector describing item. The obtained matrix R is our estimation and we could then build the recommendation system based on its values.

In the first problem, we need to implement the weighted Non-negative Matrix Factorization algorithm and perform matrix factorization on movie rating data matrix with three latent feature k values. The U,V matrices is obtained through the following least square cost function:

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

In the equation above, the goal is to minimize the least square error value. Wij is the weighted matrix value and Rij is the original data matrix value. Here is another version of the cost function which introduces the regularization term lambda, which is used to avoid singular values in the matrix manipulation:

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

In real practice, we don’t actually need to find the min of the cost function. It does not really make any difference when the cost is small enough. Therefore, we set a small threshold value and the algorithm will stop soon as the cost decrease to this threshold value. Initialize U, V to be two random matrices, we run the following equation iteratively until the cost fall under our threshold:

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

We also set a max iteration number to be 100 just in case that the cost will never be reduced under our threshold value. The least squared error for each latent features k values is displayed in the table below:

K feature value	Least Square Error (Original R, W)
10	60824.13
50	30326.82
100	17597.17

From the table, it seems LSE gets smaller as K values grow.

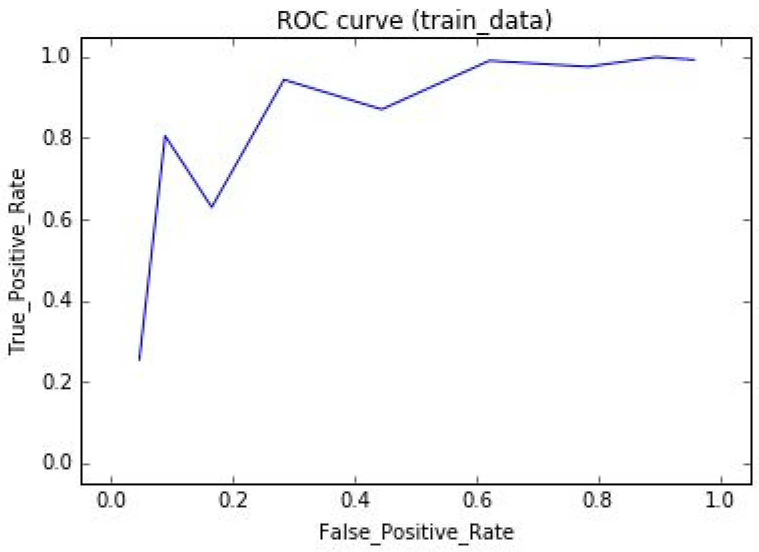
Problem II. 10-Fold Cross Validation

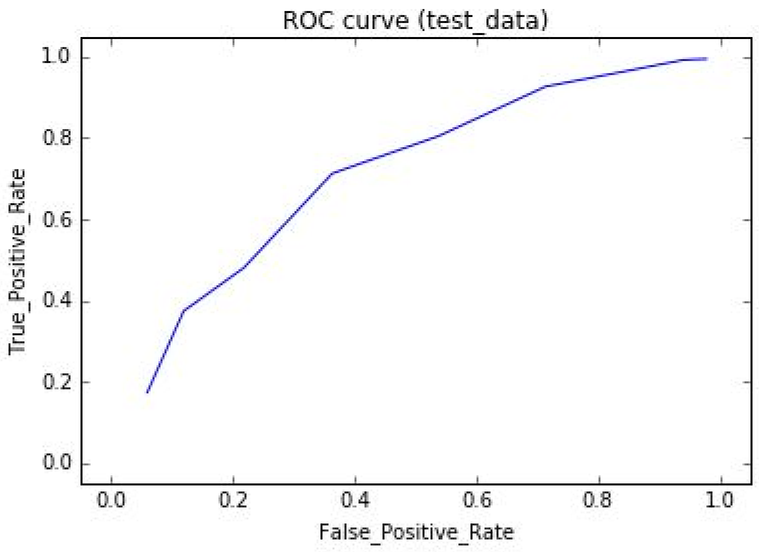
In order to perform the 10-fold cross validation using the weighted NMF function implemented in part 1, we first split the whole data into 10 parts. Each time, we will take one part as the test data and the rest as the train data so all data will be used in both train and test. Then we fit the train data into the original data matrix and let the weighted matrix only has weight equals to 1 where train data exists and the rest part is filled with 0. Next, we fit the original data matrix and the new weighted matrix into the implemented weighted nmf function and obtain the corresponding U,V. We check the average absolute value for these 10 sets and here is our result.

Average absolute value			
K value	**Average**	**Max**	**Min**
10	1.028998241624	1.101507375882	1.002511067897
50	0.925375971839	0.954643832748	0.882373262728
100	0.875446078431	0.906972350373	0.842445058665

Problem III. ROC Curve

In this part, we set the 10 threshold values from 0.5 to 5 with an even interval 0.5. For each cross validation, we collect the number of entries in the predicted train and test results where the user would like the movie for different thresholds. We also collect the number that the user like the movie based on different thresholds in the original data. Then we calculate the percentage of number of like people in actual data over that in predicted system for train data as the precision. We also calculate the percentage of number of like people in actual data over that in predicted system for test data as the recall. We finally take the average of percentages in these 10 cross fold tests and plot the precision over recall via different values of threshold for the designed recommendation system. The results are shown below.





Threshold	Precision	Recall
1	0.95714769	0.99246556
1.5	0.89573374	0.99924146
2	0.7851089	0.97523366
2.5	0.6237668	0.99018136

3	0.4475732	0.87028867
3.5	0.28685047	0.9431714
4	0.16732303	0.62917864
4.5	0.08975084	0.80570342
5	0.04691237	0.25469667

Problem IV. Regularization

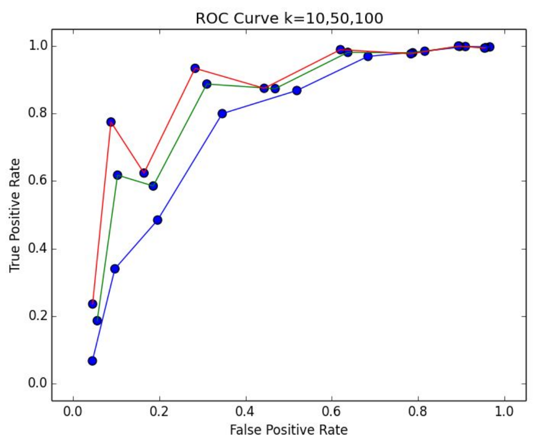
In this problem, we perform the same non-negative factorization on the data matrix as we did in problem one but reverse the role of R and W matrices in the algorithm. The total squared error value is listed as follows (the iteration value is still 100):

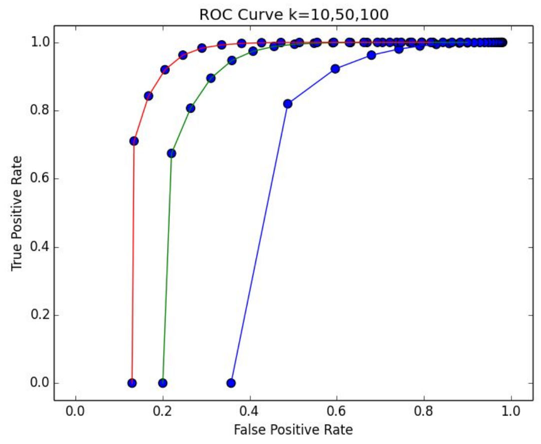
K feature value	Least Square Error (Reversed R, W)	Least Square Error (Original R, W)
10	74.65	60824.13
50	187.96	30326.82
100	147.41	17597.17

The table above shows the LSE values for original R, W matrix and the reversed case. The LSE values for the original case are the same as what we obtained from part 1. By comparing these two cases we can see that LSE values after reversing R and W matrix are much smaller than those in the original setting. This is because after we reversed two matrices, the prediction matrix obtained by multiply U with V is the prediction over matrix W, which has cell values to be either 1 or 0 depending on whether the user has rated the movie. However, in the original setting, we were using U and V to predict over matrix R, which contains cell value 0 to 5 corresponding to user's movie rating. Therefore, by switching matrices W and R in the non-negative matrix factorization, we are actually performing ***two different tasks***:

1. **To estimate user's potential rating to all rated movies (original case)**
2. **To evaluate user's likelihood to rate each movie (switching case)**

For the prediction matrix in the switching case, the closer the value in cell ij is to 1, the more likely for user i to rate movie j. Since the prediction matrix over W has most of its values less than 1, the LSE is therefore much smaller than that in the original case, with cell values that can go as high as 5. We also tell from the table that the LSE of original case is inverse proportional to K while the it is the other around in the original case.

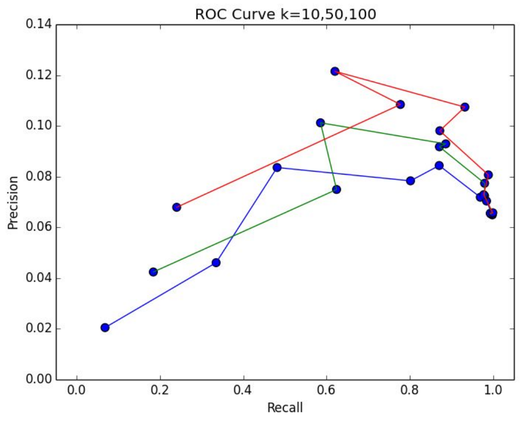


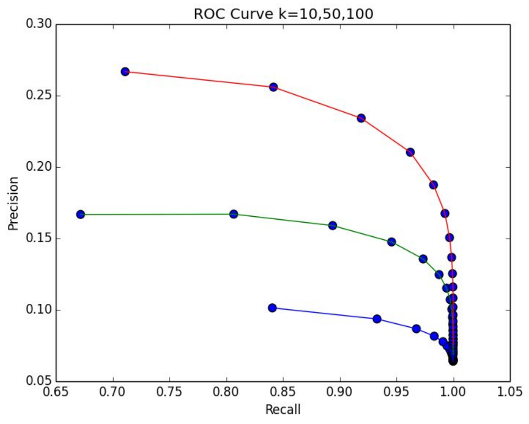


The two diagrams above reveal how k values affect the ROC curve in both cases. The graph on the left is the original case and the right is the reversed case. Note that we used different threshold ranges to draw them. For the one on the left, each point in the graph stands for a value in the threshold range of 1 to 5 with a resolution of 0.5; for the reversed case diagram, the threshold ranges from 0.8 to 1, with a resolution of 0.01. This is because we found almost all values predicted in the reversed case are in the range of 0.8 to 1. Blue, green and red curves in above diagram respectively corresponds to k values of 10, 50 and 100.

Clear from the diagram we can see the higher the K value is, the more the ROC curve shifts to the left. This means the ratio of True Positive Rate over False Positive Rate grows as k grows. In other words, we are making more accurate prediction with bigger k values.

The two diagrams below are PR (Precision vs. Recall) curves of the original (left) and reversed matrix (right) cases:





Blue, green and red curves in above diagram respectively corresponds to k values of 10, 50 and 100 just as before. Precision values of both cases are quite low as indicated above.

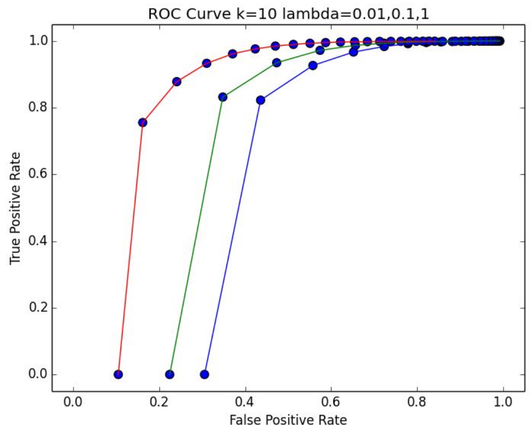
In order to avoid singular values, we modified the cost nmf algorithm by adding a regularization term lambda. We chooses values of lambda to be 0.01, 0.1 and 1. For each k value we perform the regularized algorithm for each lambda value and then pick the one with the best least square error.

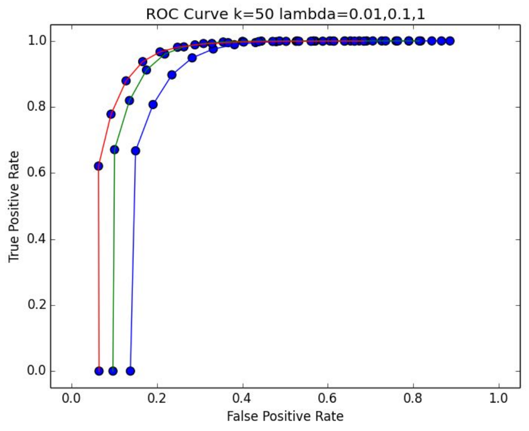
R and W Matrix reversed case:

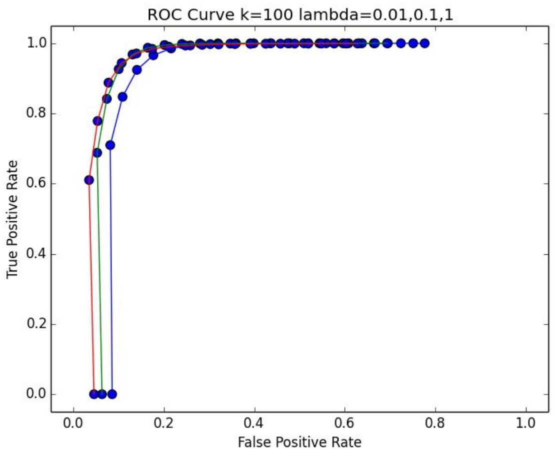
k\lambda	0.01	0.1	1
10	62.38	64.51	96.67
50	185.96	164.3	178.36
100	146.6	141.2	168.34

Based on the table above, there seems to be no obvious pattern of how LSE changes with k or lambda. The best LSE value is 62.38 and it happens when k is equal to 10 and lambda is equal to 0.01.

The following three diagrams show the ROC curve (True positive rate vs False positive rate) for three lambda values after we reverse weight and R matrix:

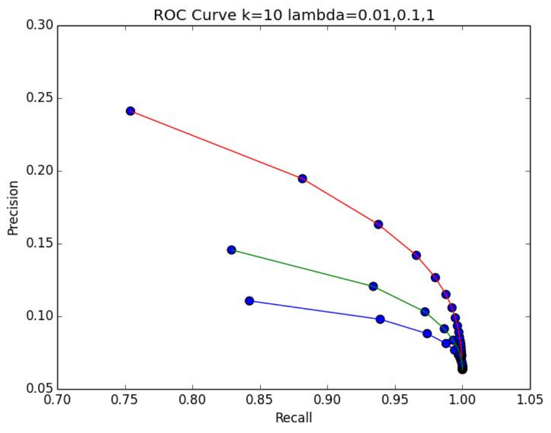


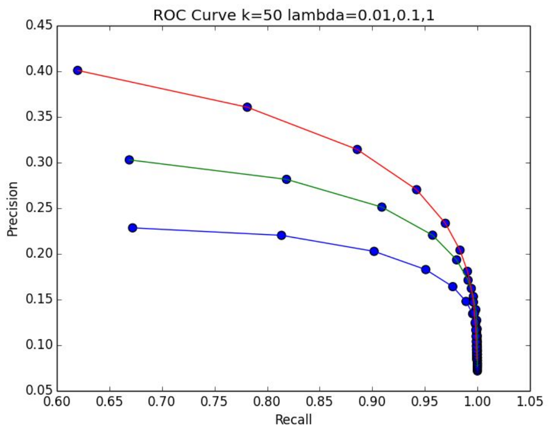


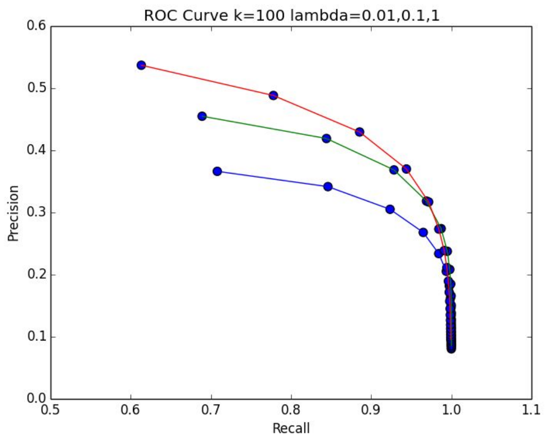


For the three diagrams above, the threshold range is still choose to be within 0.8 to 1 with 0.01 resolution. In each diagram, red, green and blue curve represents the lambda value of 1, 0.1 and 0.01. From the diagram, the rule we found previously that 'larger k value shift curve to the left' still holds. The introduction of lambda value also shifts the curve. It seems that larger lambda value also cause the ROC curve to shift to the left. However the effect of lambda tends to decrease when the k value grows. Furthermore, compared with the ROC curve obtained earlier, lambda value also change the smoothness of the curve. The larger the lambda value, the more smooth ROC curves become.

The following three diagrams show the PR (Precision vs Recall) curves:







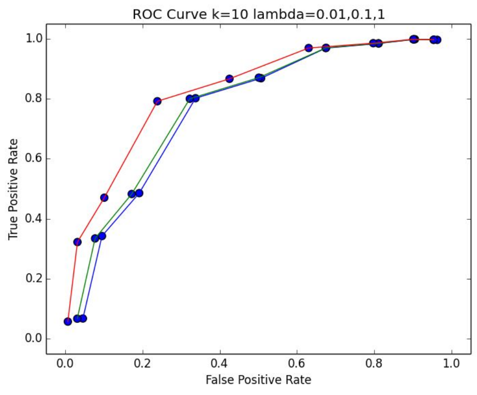
In each diagram, red, green and blue curve represents the lambda value of 1, 0.1 and 0.01. Clearly, the higher the lambda and K values, the higher precision values it gets.

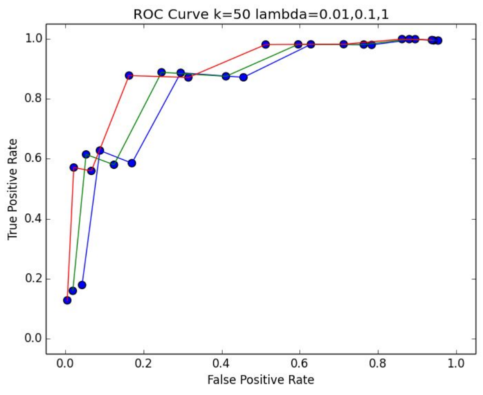
R and W matrix not reversed case:

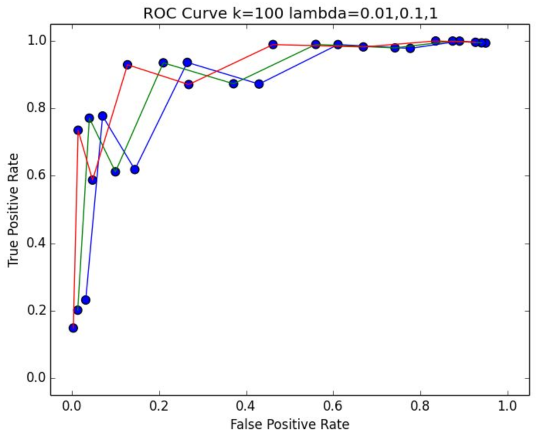
k\lambda	0.01	0.1	1
10	60963.77	60440.27	60874.84
50	30163.86	30751.7	31998.41
100	17752.42	17862.97	19433.28

In the non-reverse case, from the table above, it seems that lambda value does not have significant effect on LSE except for lambda =1 in the k=100 case. The rule that LSE decrease with k values still holds.

The following three diagrams show the ROC curve for three lambda values if we do not reverse weight and R matrix:

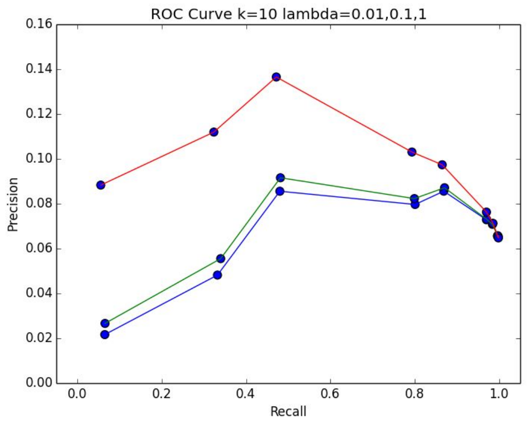


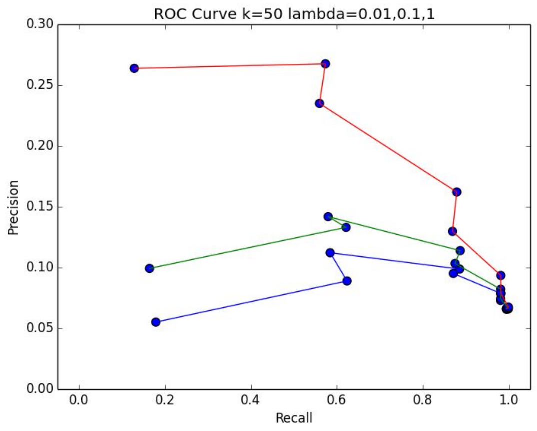


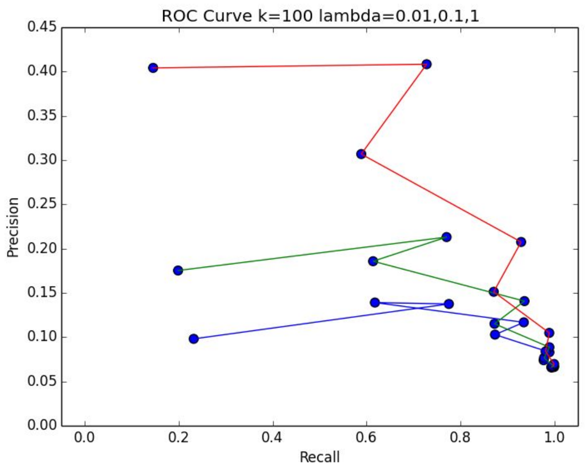


For above three diagrams, same as the previous section, the red, green and blue curve corresponds to the lambda value of 1, 0.1 and 0.01. Larger lambda value still shifts curves to the left but not as much as in the reversed case. Also, the curve tends to be smoother with smaller k values.

The following three diagrams show the PR curve for three lambda values if we do not reverse weight and R matrix:



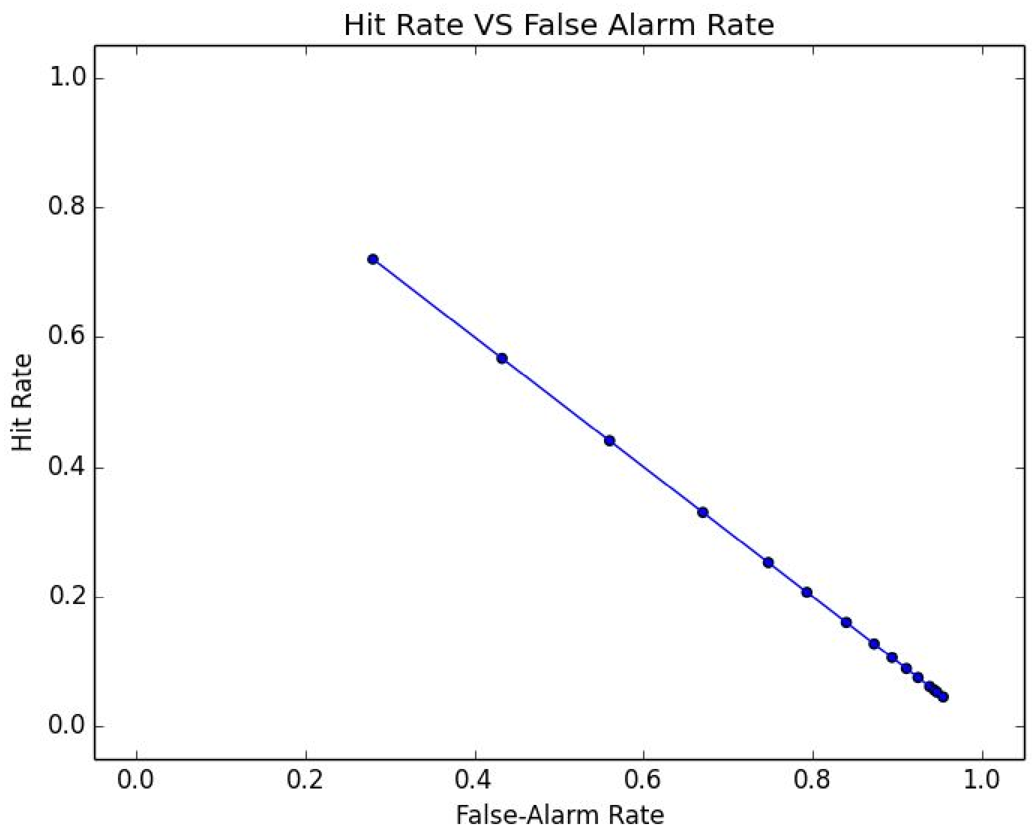




Same as the before, red, green and blue curve corresponds to the lambda value of 1, 0.1 and 0.01. The observation from the reverse case still holds. The higher lambda and k values, the higher precision we could achieve.

Problem V. Recommendation System Creation

We then try to provide the users a top rated movie list based on their interest on other movies. We use the NMF method we use for part I and predict the user data with 10 cross validation. In this problem, we use the weight of train data as our input X for NMF and use the R matrix as our weighted matrix for NMF. After we get the predicted R, we will then iterate on each user on the test group. on each user in our test group, we sorted its favourite movie on the predicted R matrix and get the top L movies we will suggest for the test users(those movies has to have a rating by the same user). Then, we will compare the predicted rating and the actual rating by using the threshold value = 3 to see if user really likes the movie or not. Then we iterate on L as well and we can get the graph for L from 1 to 15



hitRate for L = 5 is: 0.252386002121
falseRate for L = 5 is: 0.747613997879