EE239AS

Project 3

Collaborative Filtering

Tongtong Xu 004592767
Dongdong Cai 304587528
Jiajun Zhu 904618107

INTRODUCTION

In order to implement a recommendation system from the matrix R whose row represent the user-id and column represent the movie-id. We need to obtain the matrix for user-preference(U) and feature-movie(V) which have much lower dimension than R. We can factorize R = UV using alternating least square algorithm. That is, randomly pick U first and calculating the V such that minimize $||R - UV||_F^2$, then with calculated V, update U to minimize $||R - UV||_F^2$. Keep doing this until U and V converges. The algorithm can be easily implemented using multiple programming languages. Here we use Matlab to do this since Matlab is easier to use in terms of matrix calculation. In this project, we would use 100k dataset from MovieLens which contains 943 users and 1682 movies.

PART 1

When using the ALS algorithm, we have to consider the case when rating value is missing since users can not always rate all movies. Here we use a weight matrix(W) when we calculating squared error. The W contains 1 where we have known data and 0 where we data is missing. Now, we can perform out ALS algorithm with wnmfrule function in Matrix Factorization Toolbox. Here we set the the number of latent features as 10, 50, 100. The results are as following.

k	10	50	100
squared error	5.462734575256868e+04	1.959206455358187e+04	6.203894007041163e+03

As we can see from the result above, we can improve the result of our algorithm by increasing the number of latent features. This is a intuitive result since we can obviously get better result if we know the preferences of users and features and movies more.

PART 2

In this part, in order to do 10-fold-cross-validation, we just shuffle the 100k raw data in iteration i we get data number (i - 1) * 10000 + 1 ~ i * 10000 in shuffled raw data as our testing set and the rest as our training set. Notice that we have to set the test entities to 0 in the weight matrix. The results are as below.

	min_error	max_error	mean_error
k = 10	0.7880	0.8279	0.8021
k = 50	0.9403	0.8210	0.8807
k = 100	0.9586	1.0039	0.9720

Note that due to some issues, We believe it is because the wnmfrule function and we still can not fix it, sometimes our program would produce extremely big result. So, sometimes you should run the program multiple times to get a reasonable result.

PART 3

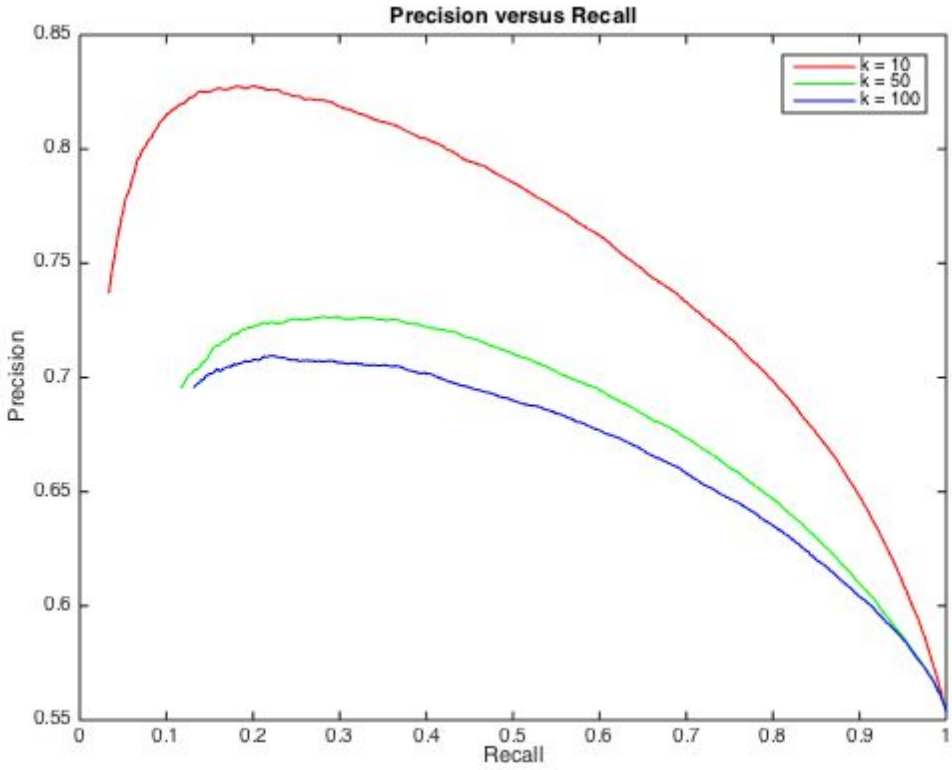
To plot the recall vs precision curve, we can use the data we predicted in part 2 which can save us a lot of time. We use the following logic to calculate the recall and precision value.

```
if predictedR{1, m}(user, movie) > thre3(k)
        precisionpre3 = precisionpre3 + 1;
        if R(user, movie) > 3
                precisionact3 = precisionact3 + 1;
        end
end
if R(user, movie) > 3
         recallact3 = recallact3 + 1;
        if predictedR{1, m}(user, movie) > thre3(k)
                recallpre3 = recallpre3 + 1;
        end
end
precision3(m, k) = precision3(m, k) + precisionact3 / precisionpre3;
recall3(m, k) = recall3(m, k) + recallpre3 / recallact3;
```

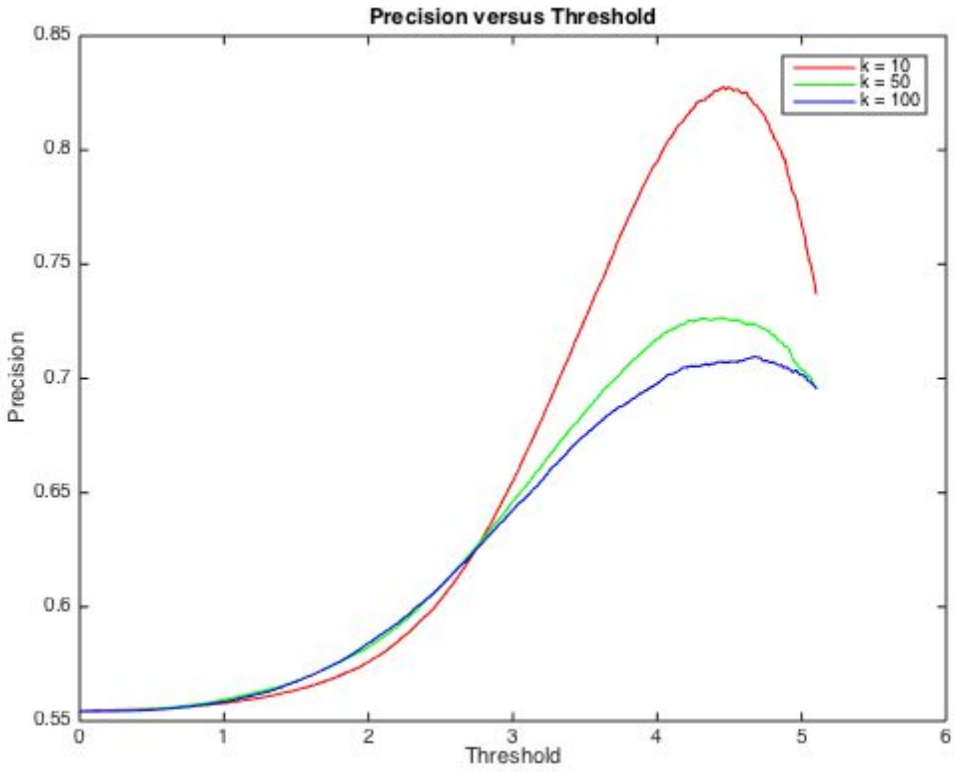
That is, precision means the percentage that user actually like the movie among prediction and recall mean the percentage that the system recommend to the user among the movies the user actually like.

What we can learn from our result is that we can not get good precision and recall at same time, so maybe we need a new algorithm. I believe this is what we are going to do in part 4.

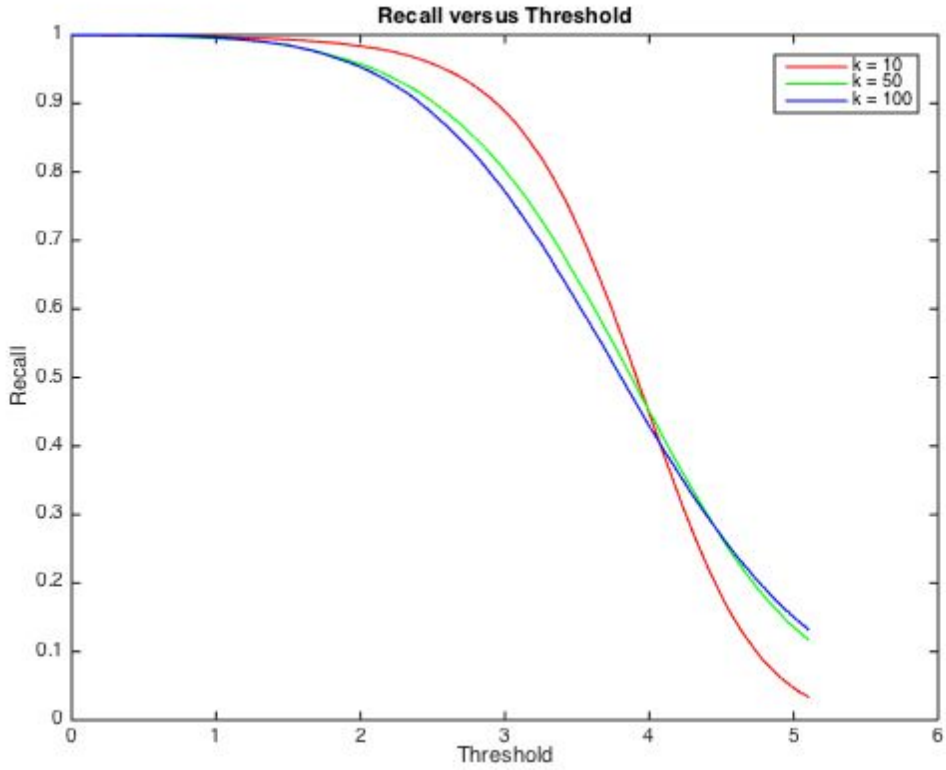
The following graph is the recall vs precision curve we plot.



The following graph is the threshold vs precision curve we plot.



The following graph is the threshold vs recall curve we plot.



PART 4

It is easy to implement the idea in part 4 by just switching the weight matrix and R. The following chart is the squared error we get.

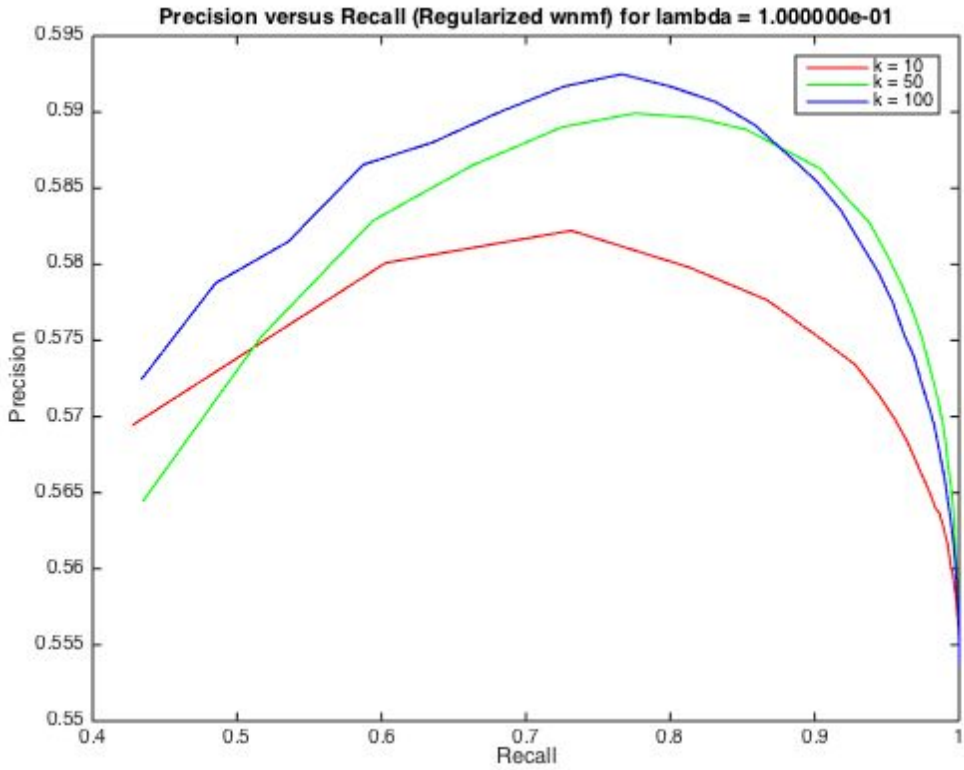
k	1	50	100
squared error	1.037805019475700	8.699112004538840	22.999207050405385

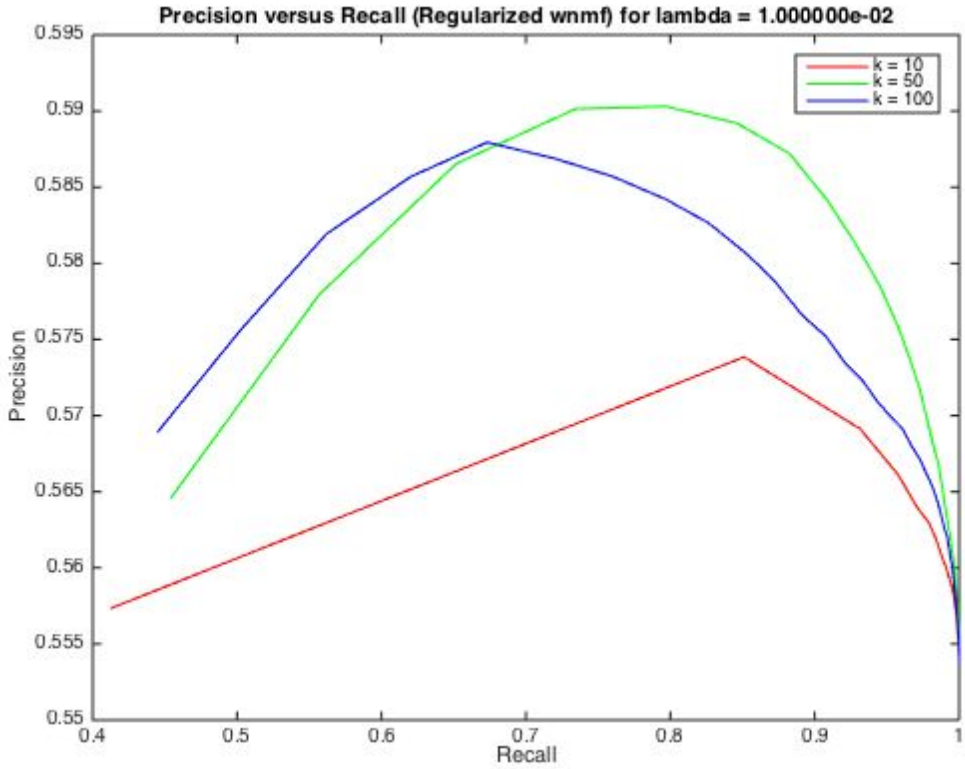
To implement the modified version of cost function in wnmfrule function we have. We use different R, U and W to calculate V and different R, V ,W to calculate V. Here are the details.

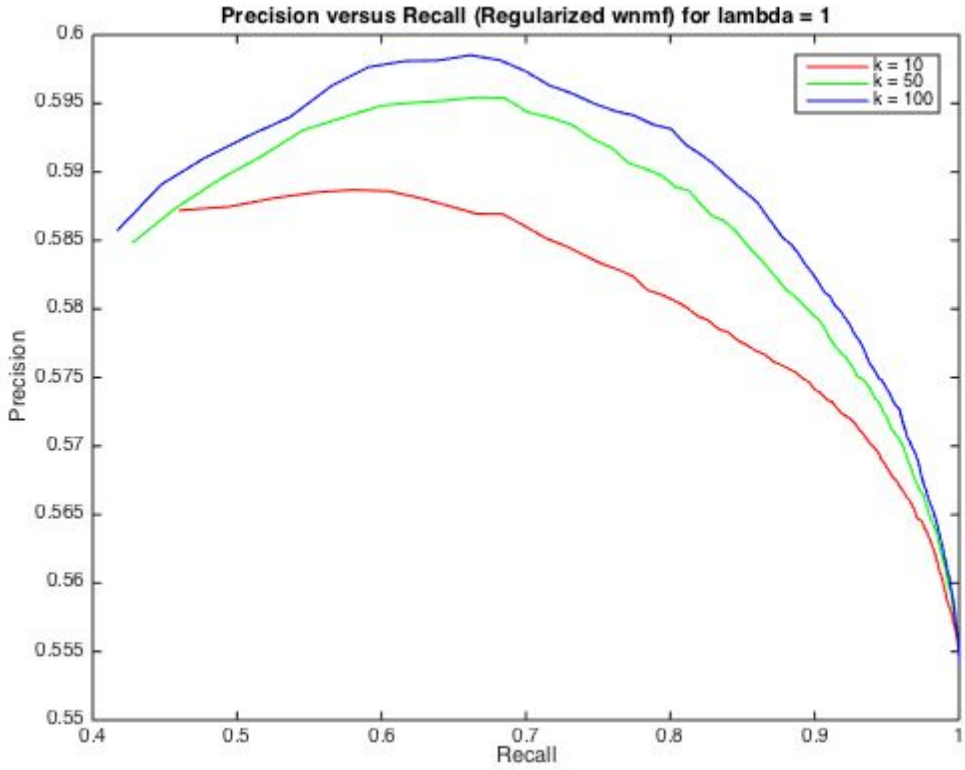
```
Wt = [W'; ones(k, r)];
Wnt = [W; ones(k, c)];
R0t = [X';zeros(k, r)];
R0 = [X; zeros(k, c)];
Vlt = [Y';sqrt(lambda) * eye(k, k)];
Ue = [A;sqrt(lambda) * eye(k, k)];
```

Then we factorize R0 = Ue * V and R0t = Vlt * U' with Wnt and Wt as the weight matrix, it is easy to prove we can minimize new cost function through doing the transforms above.

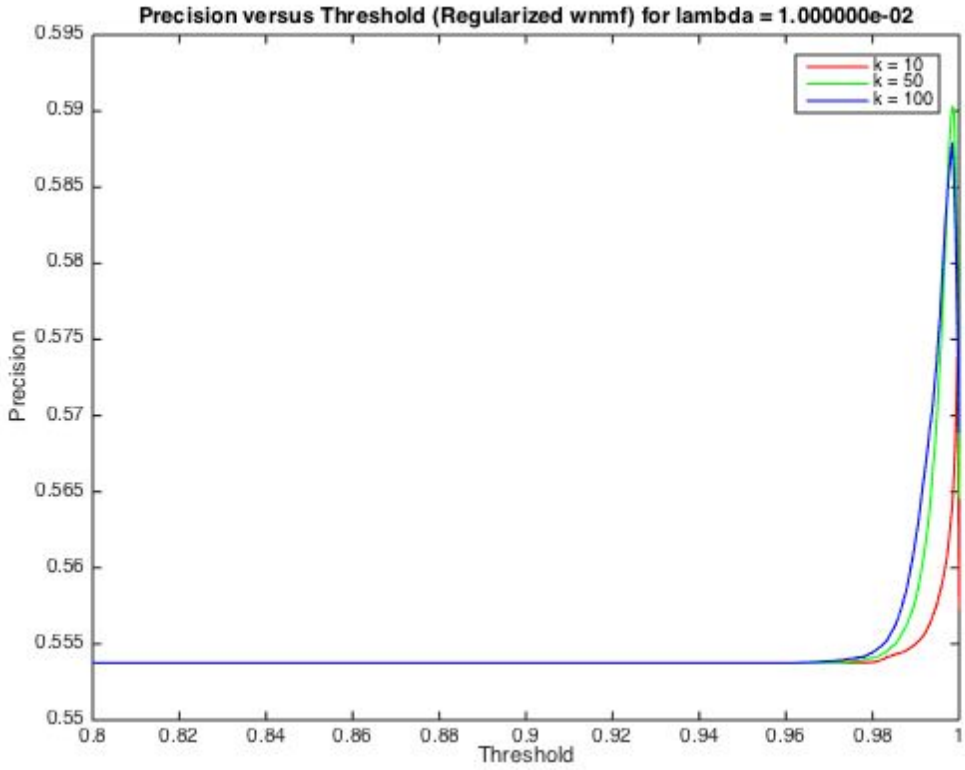
The following graphs are the recall vs precision curve with new cost function.

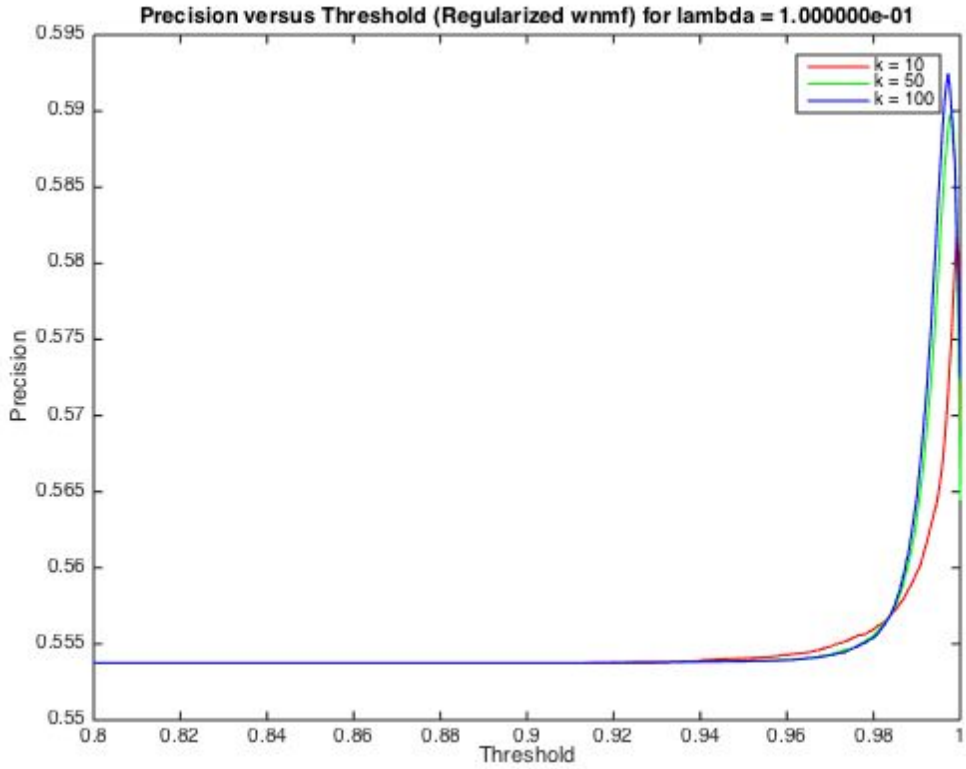


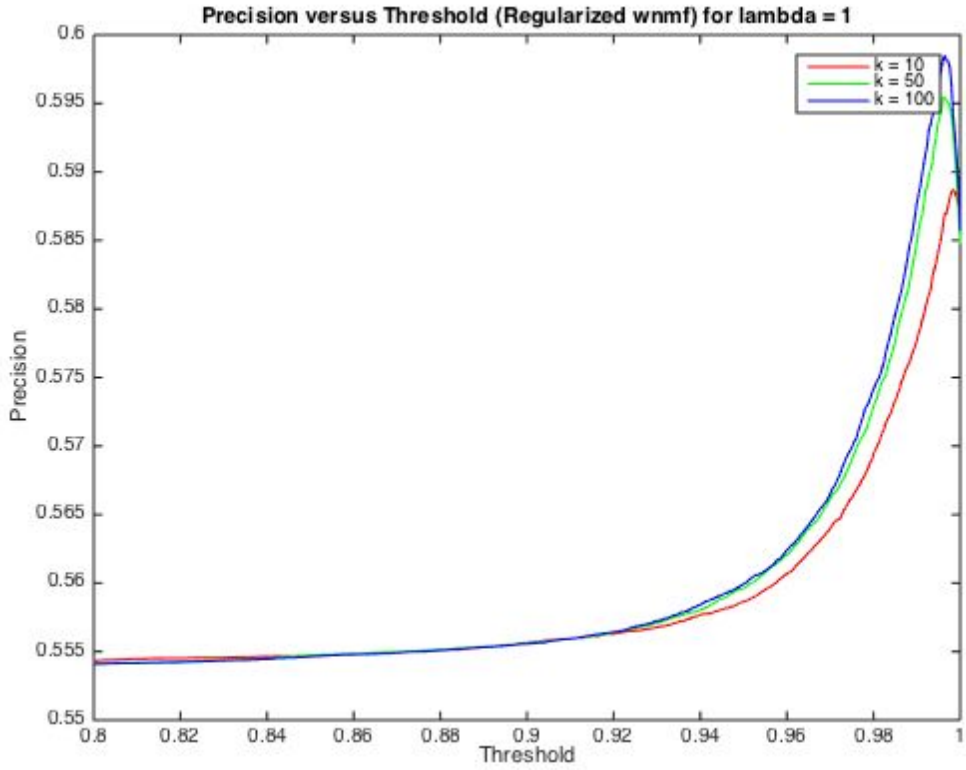

Precision versus Recall (Regularized wnmf) for lambda = 1.000000e-02
k = 10
k = 50
k = 100
Precision
Recall
0.595
0.59
0.585
0.58
0.575
0.57
0.565
0.56
0.555
0.55
0.4
0.5
0.6
0.7
0.8
0.9
1


Precision versus Recall (Regularized wnmf) for lambda = 1
k = 10
k = 50
k = 100
Precision
Recall
0.6
0.595
0.59
0.585
0.58
0.575
0.57
0.565
0.56
0.555
0.55
0.4
0.5
0.6
0.7
0.8
0.9
1

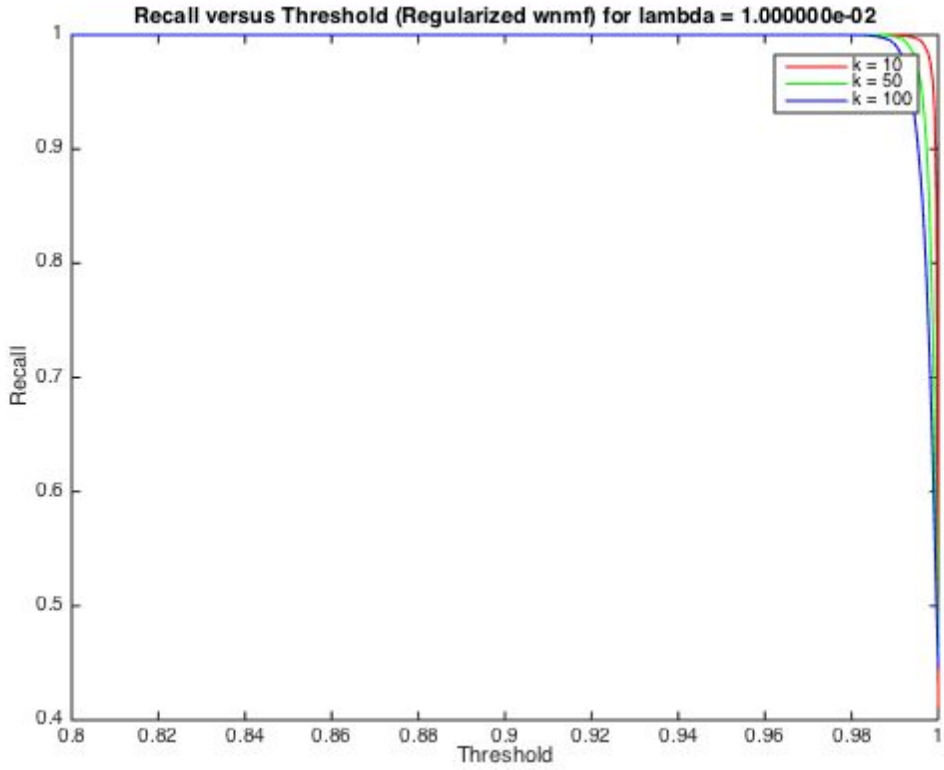
The following graphs are the threshold vs precision curve with new cost function.

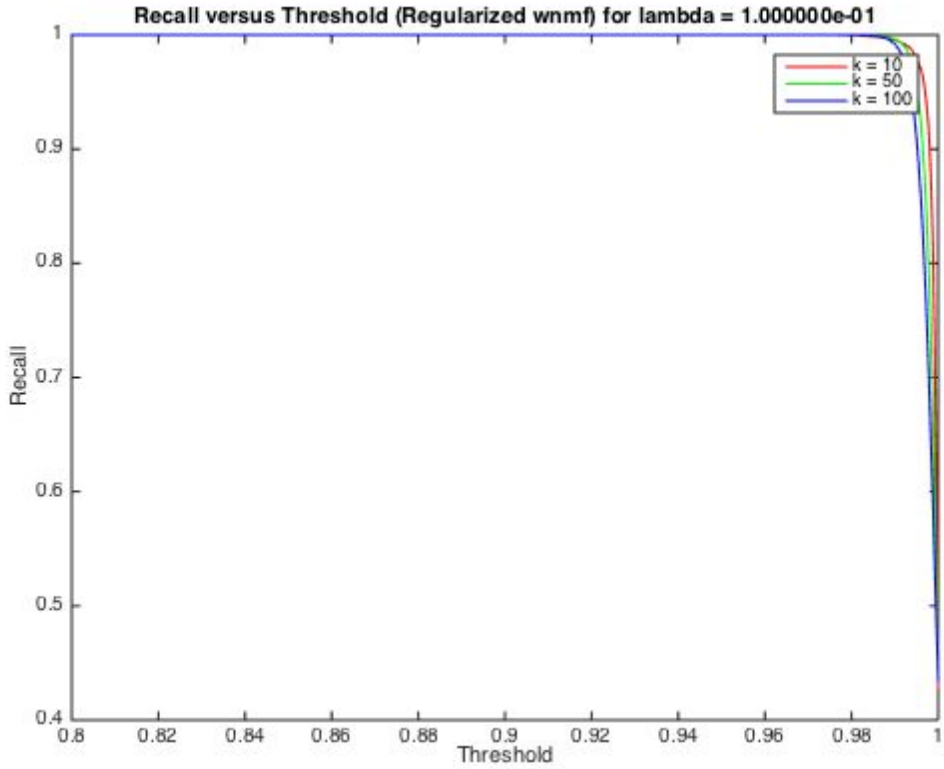


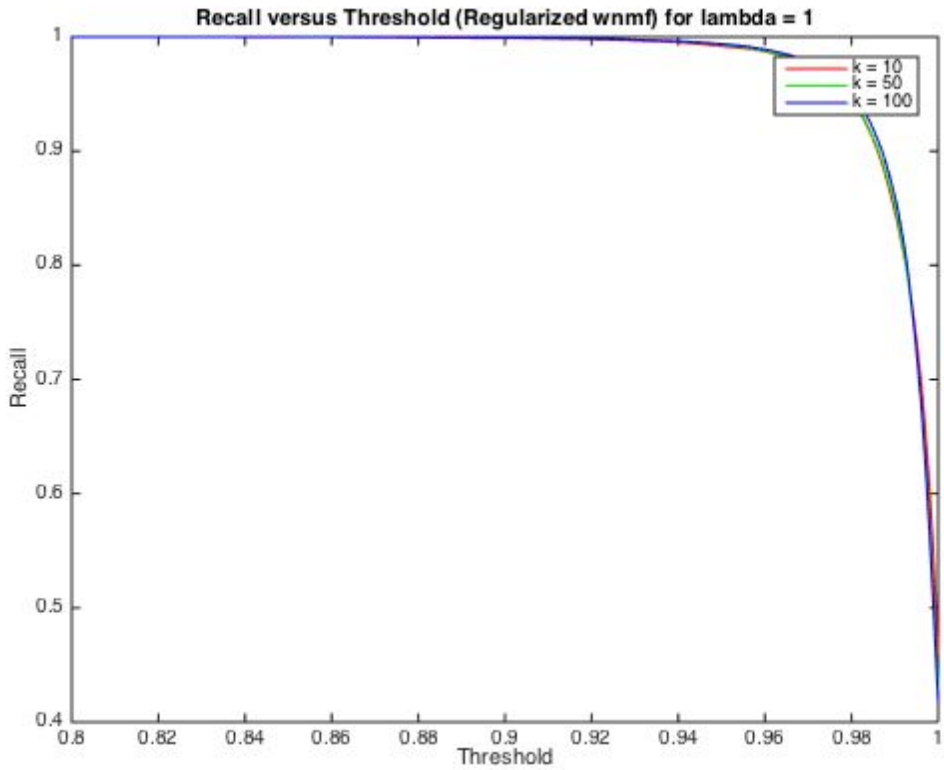

Precision versus Threshold (Regularized wnmf) for lambda = 1.000000e-01
k = 10
k = 50
k = 100
Precision
Threshold
0.595
0.59
0.585
0.58
0.575
0.57
0.565
0.56
0.555
0.55
0.8
0.82
0.84
0.86
0.88
0.9
0.92
0.94
0.96
0.98
1


Precision versus Threshold (Regularized wnmf) for lambda = 1
k = 10
k = 50
k = 100
Precision
Threshold
0.6
0.595
0.59
0.585
0.58
0.575
0.57
0.565
0.56
0.555
0.55
0.8
0.82
0.84
0.86
0.88
0.9
0.92
0.94
0.96
0.98
1

The following graphs are the threshold vs recall curve with new cost function.







We can learn from the graphs above that the precision of the algorithm is improving with the increasing of the lambda but it is still not good enough with lambda = 1. We tried lambda = 10 and get a precision about 70% but the algorithm would become extremely

slow. Another thing we find out is that we should set the threshold at about 0.985 to get a good precision and recall when k = 100 and lambda = 1. So, we would use these parameters to build our recommendation system in part 5, although in our source code we tried k = 10, 50, 100.

PART 5

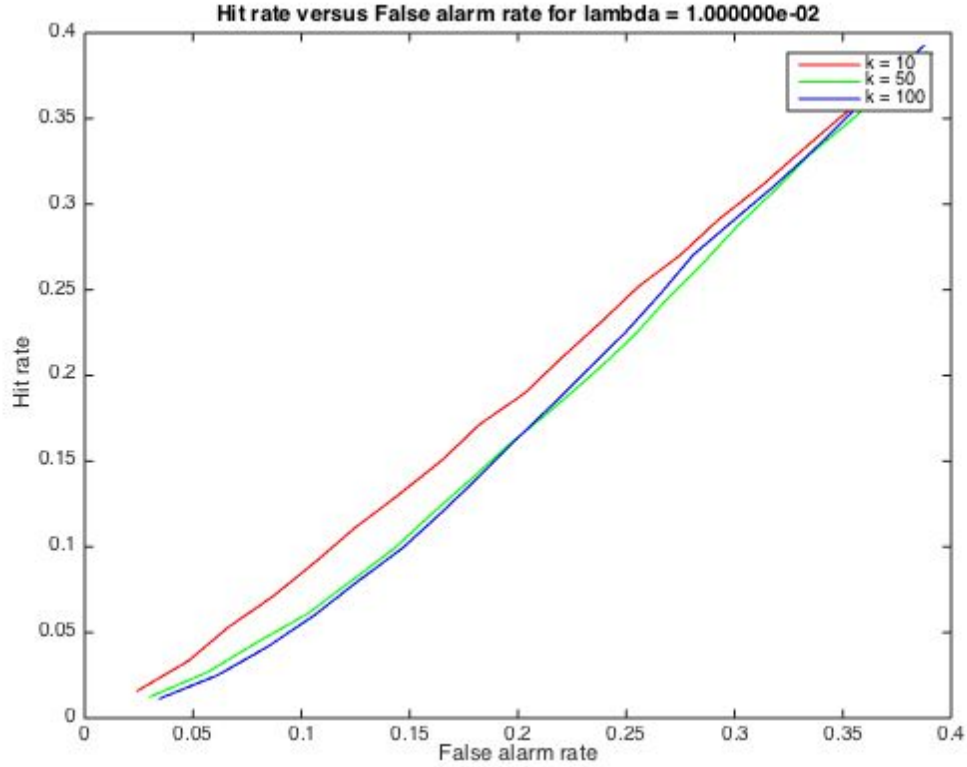
From the result above in part 4, in order to get optimal result(best precision and recall) to set our lambda as 1 and k as 100 to build our recommendation system. In this part, we sort every row of the predicted matrix first and pick top L movies for each user and find the fraction of the test movies liked by the users are suggested by our system(Hit rate) and the fraction of the test movies not actually liked by the user, are suggested by our system(False alarm rate). The following code is the logic of how we get our hit rate and false alarm rate.

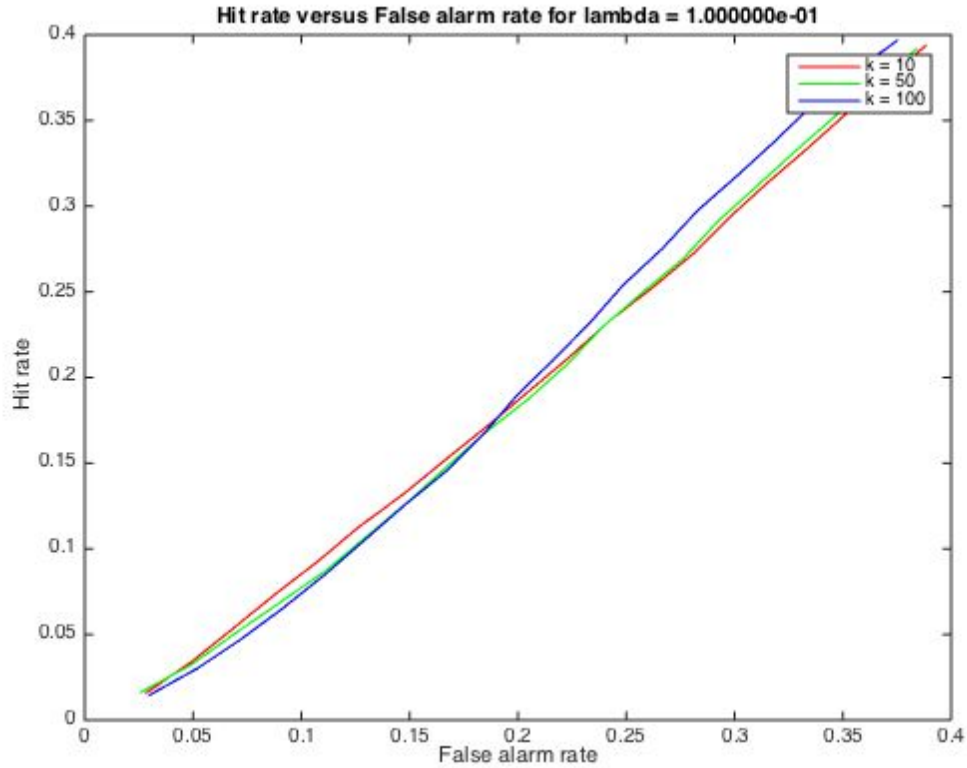
```
if R(j, pretopL(k)) > 3
        actlike = actlike + 1;
end
if R(j, pretopL(k)) <= 3
        dislike = dislike + 1;
end
hitrate = hitrate + actlike / liketotal;
falserate = falserate + dislike / disliketotal;
```

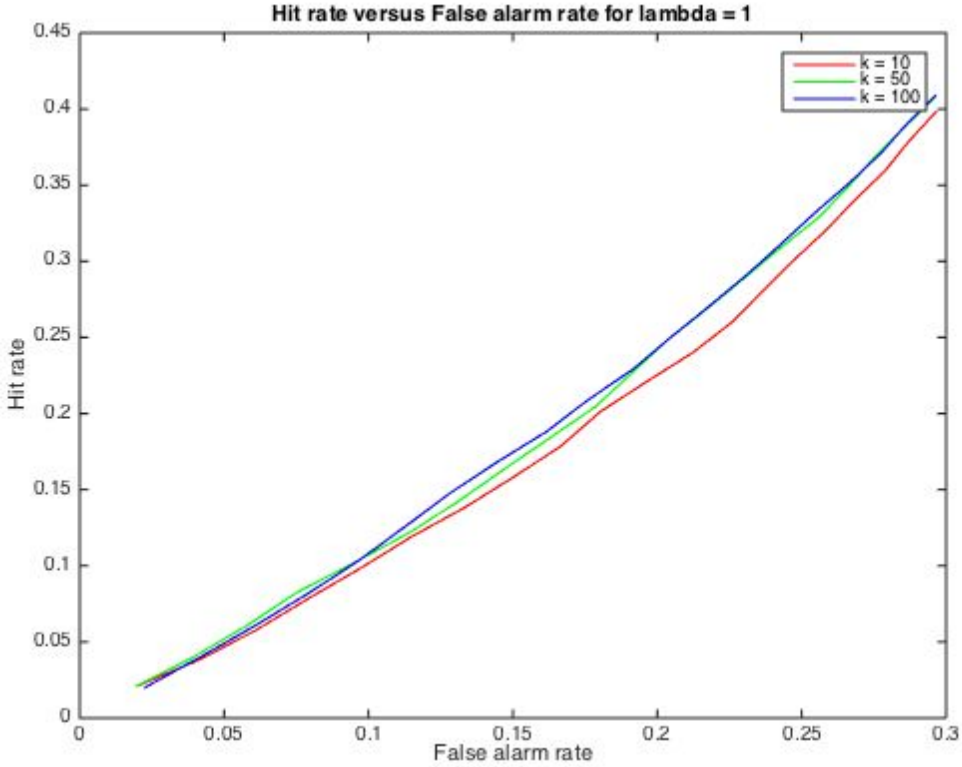
Here is the average precision(fraction of movie actually liked by user among 5) for L = 5

	lambda = 0.01	lambda = 0.1	lambda = 1
k = 10	0.7752	0.7785	0.8142
k = 50	0.7236	0.7786	0.7491
k = 100	0.7200	0.8142	0.8104

The followings are the result graphs. Although we choose lambda = 1 and k = 100 to implement our system, we also plotted the graph for different lambda and k for comparison.


Hit rate versus False alarm rate for lambda = 1.000000e-02
k = 10
k = 50
k = 100
Hit rate
False alarm rate
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4


Hit rate versus False alarm rate for lambda = 1.000000e-01
k = 10
k = 50
k = 100
Hit rate
False alarm rate
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4



As it can be seen from the graphs above, the system's hit rate is still not good enough. We believe we can solve this problem by increasing lambda to a number that is greater than 1 which would make the program slower.