Project 3
Collaborative Filtering

Donna Branchevsky
UID: 404473772

Pavan Holur
UID: 204403134

Megan Williams
UID: 104478182

Tadi Ravi Teja Reddy
UID: 505227246

University of California, Los Angeles

Winter 2019

Question 1

We first analyzed the data set provided, by adding the data folder to our default path and examining the **ratings.csv** file. The file consists of a table with **UserID**, **MovieID**, **Rating**, and **Timestamp** rows. Importing these entries into our script was done using the Pandas library. The output was a list of tuples. Please find the code below:

```
def retrieve_data():
    file_path = './ml-latest-small/ratings.csv'
    reader = Reader(line_format='user item rating timestamp', sep=',',
    rating_scale=(0.5, 5), skip_lines=1)
    data = Dataset.load_from_file(file_path, reader=reader)
    return data
```

The tuples can now be transformed into a matrix with each row representing the users and each column representing a movie. We assume the default rating to be 0. This is legal s the lowest rating provided in the set is 0.5. The sparsity was calculated with the given formula and the sparsity $\mathbf{S = 0.017}$.

Question 2

Following the design of matrix R and the calculation of sparsity S, we proceed to bin the ratings into 0.5 wide bins. Note that the ratings are in fact **discrete** and of the values $[0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]$. So we may bin the ratings accordingly. Note: **It is not good practice to bin ratings such that all values fall on the edge of the bins.** Therefore our bins are split as follows: **[0.25, 0.75, 1.25, 1.75, 2.25, 2.75, 3.25, 3.75, 4.25, 4.75, 5.25]**

The histogram is given below. Note that there are more elements with rating 4 than any other rating. Also the ratings seem to skew slightly towards the higher ones. Inferences include a. More people rate movies that they find good and b. More people vote integer ratings.

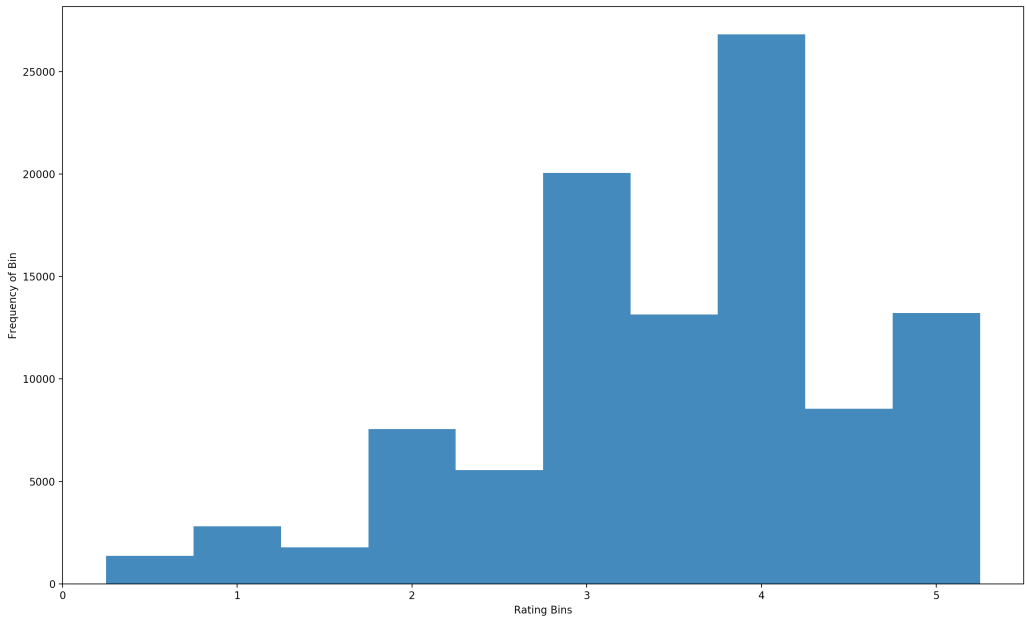

Figure 1: Rating Histogram per movie (discrete bins)

Question 3

The matrix is then sorted along rows and columns based on number of entries for users and number of entries for movies respectively. **This is the matrix** R. This matrix now allows us to define the user labels and movie labels for each movie according to the specifications. The following code snippet shows the sorting process on R:

```
R = R[:,np.sum((R>0.0).astype(int), axis = 0).argsort()[::-1]]
R = R[np.sum((R>0.0).astype(int), axis = 1).argsort()[::-1],:]
```

With the revised matrix, we can plot the number of entries vs. movie. Provided our sorting function above is correct, we hope to find a decreasing function in number of ratings with movie.

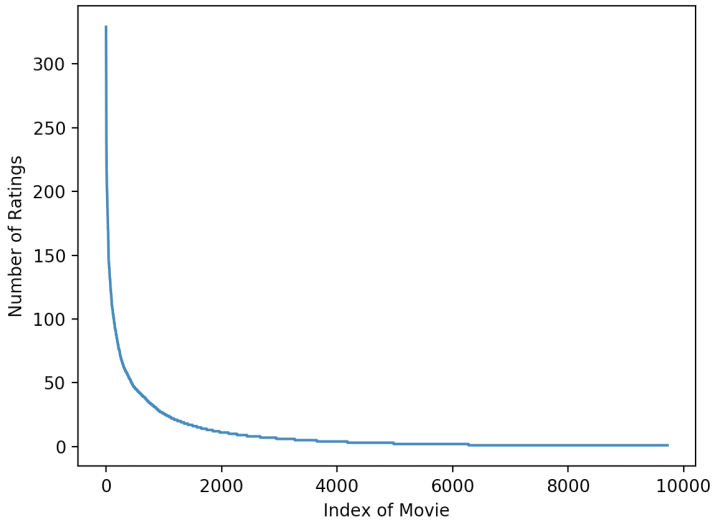

Figure 2: Number of Ratings per movie index

Question 4

Like Question 3 we now proceed to replicate the process with Users on the x-axis. The number of ratings is once again expected to be decreasing based on our sorting algorithm described in the previous section. The graph is provided below.

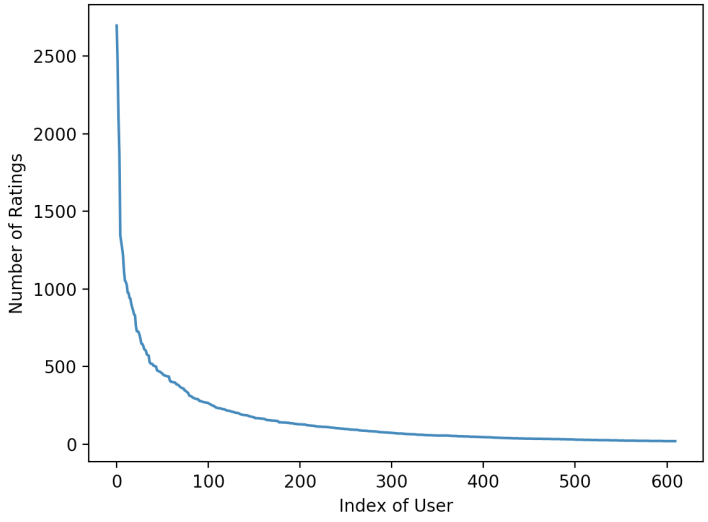

Figure 3: Number of Ratings per user index

Question 5

The resulting graph verifies that there are 9700 movies in the data set and is exponentially decaying, implying the following:

1. The ratings for movies is not uniform. Some movies are more frequently rated than others; as a result, sparsely rated movies will have a poorer success rate in the recommendation.

2. Further if our recommendation model predicted on quality of rating and number of ratings, movies with few but healthy evaluations might be a fringe recommendation in our model further moving them down the curve, despite being "good" movies. (This is in exchange to the "tried-and-tested" movies.)

Thus there are concerns to implement an "exploration factor". A good recommendation system would have to create: (a graph that is as "less steep" as possible)

1. Sufficient number of movies that are rated in high frequency

2. A rating that correlates to this frequency

Question 6

We now proceed to find the variance of each movie, or in other words, how polarizing the movie is among the audience who viewed it. The graph according to the labels defined in part 3, is given below. Also the variances are computed from matrix R as follows:

```
mean = np.sum(R, axis = 0)/number_of_entries
ssq = np.sum(R**2, axis = 0)/number_of_entries
fin = ssq - mean**2
```

The variance of most movies is low. This indicates that viewers in general have a good understanding of what others think of the movie and rate similarly. (This is why collaborative filtering is a suitable strategy.)

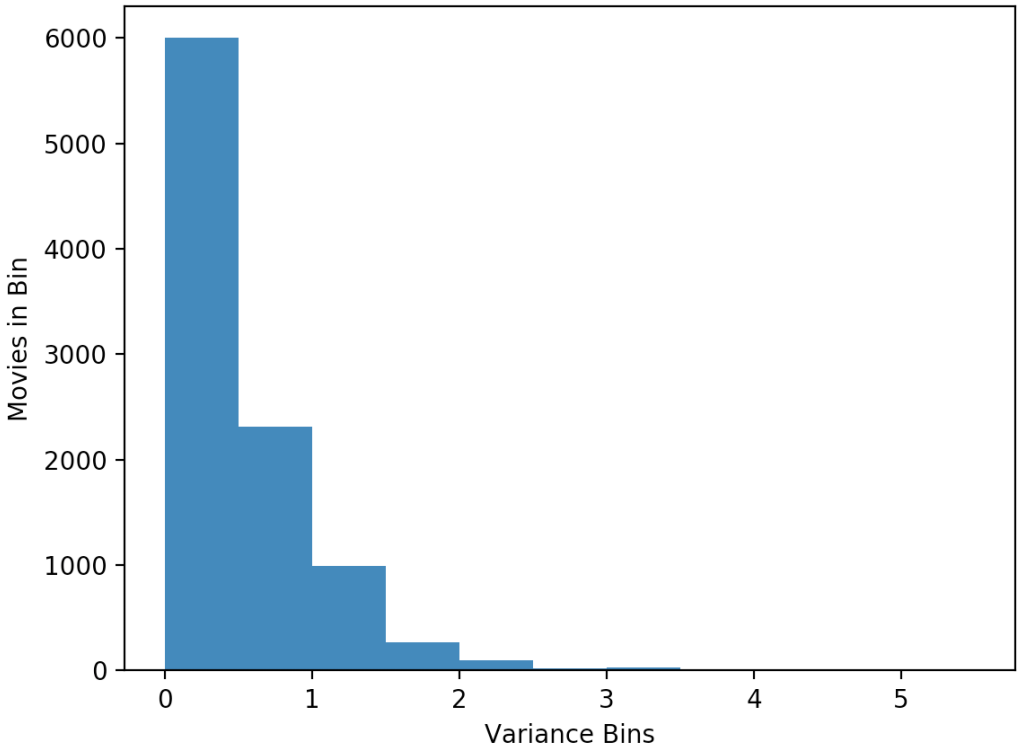

Figure 4: Variance per movie index

Question 7

Pearson-correlation coefficient between users u and v, denoted by Pearson(u, v), it captures the similarity between the rating vectors of users u and v. Before stating the formula for computing Pearson(u, v), let's first introduce some notation:

1. I_u : Set of item indices for which ratings have been specified by user u
2. I_v : Set of item indices for which ratings have been specified by user v
3. μ_u : Mean rating for user u computed using her specified ratings
4. r_{uk}: Rating of user u for item k

The mean μ_u in terms of I_u and r_{uk}, will be given by:

$$\mu_u = \frac{\sum_{k \in I_u} r_{uk}}{|I_u|}$$

Question 8

$I_u \cap I_v$ denotes the set of items for which both the users, u and v, have given rating. It could be a empty set $\emptyset$, if they have rated completely different set of items. Since the matrix is sparse it's very likely that $I_u \cap I_v$ is $\emptyset$ in a lot of cases.

Question 9

Since different user's have different way's of rating movies, we center the mean of raw ratings for generalization. For example, user-i prefers giving out high scores like 3 to 5 for most of the movies, whereas user-j prefers giving 1 to 4 for movies. This variation in style of rating movies by different users, causes bias in the rating pattern (might cause trouble while trying to use raw data for predictions), to normalize these effects we perform mean centering.

Question 10

In this section we implement a k-NN collaborative filter for predicting ratings of the movies using MovieLens dataset. We use 10-fold cross-validation and sweep k (number of neighbors) from 2 to 100 in step sizes of 2, and for each k compute the average RMSE (Root mean square error) and average MAE (Mean absolute error) obtained by averaging the RMSE and MAE across all 10 folds, then we plot the average RMSE and MAE against k.

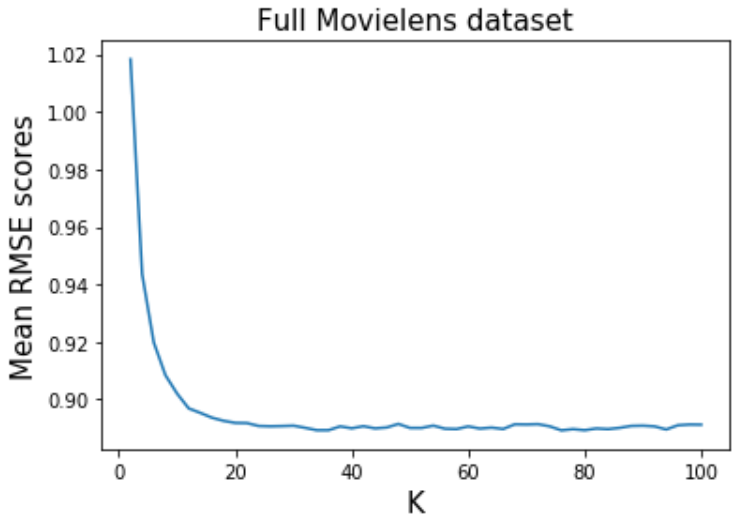

Figure 5: Plot of average RMSE against k

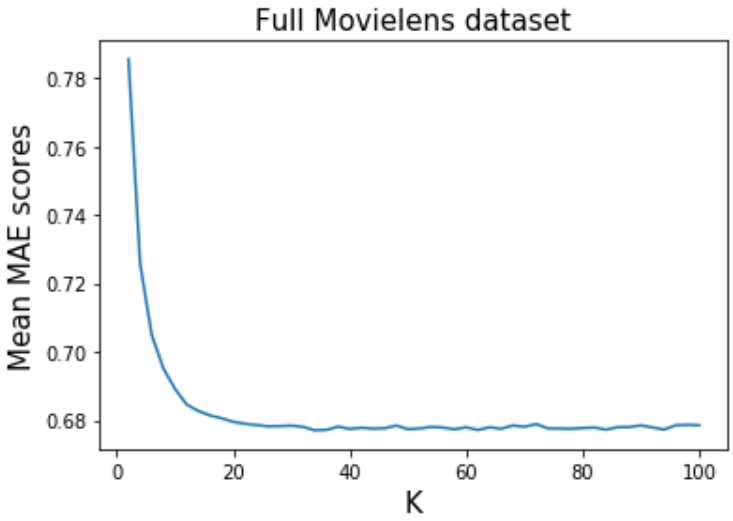

Figure 6: Plot of average MAE against k

Question 11

From the previous two plots, we observe that the **minimum value of k for RMSE and MAE to be around** 20 **and** 20 **with values 0.89 and 0.68 respectively**. We observe form the plots that minimum k for both the plots corresponds to the point where the average values of RMSE and MAE converge to a steady state.

Note: In the context of this question, the term 'minimum k' means that increasing k above the minimum value would not result in a significant decrease in average RMSE or average MAE.

Question 12 - 14

In this section, we will analyze the performance of the k-NN collaborative filter in predicting the ratings of the movies in the trimmed test set. We consider the following trimmings:

1. Popular movie trimming
2. Unpopular movie trimming
3. High variance movie trimming

In **Popular movie trimming**, we trim the test set to contain movies that have received more than 2 ratings. To be specific, if a movie in the test set has received

less than or equal to 2 ratings in the entire dataset then we delete that movie from the test set and do not predict the rating of that movie using the trained filter. The plot for this dataset is similar to the one obtained for the complete dataset, this is expected since there are majority of the movies have a high number of ratings.

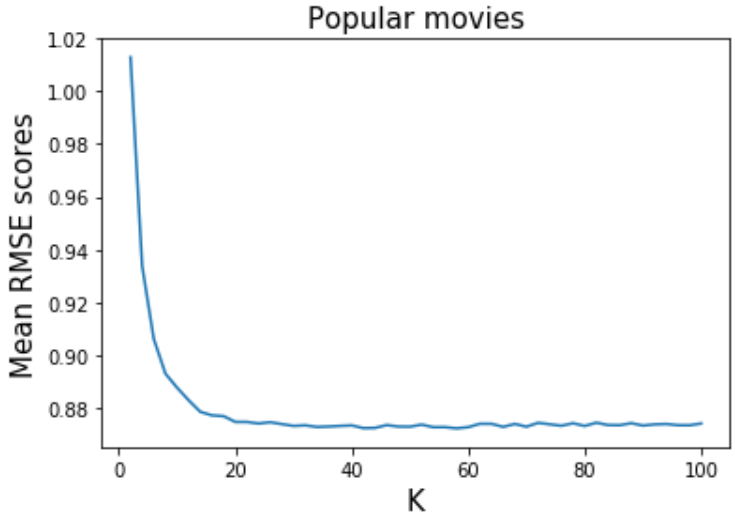

Figure 7: Plot of average RMSE against k for Popular movie trimming

Next, we trim the test set to contain movies that have received less than or equal to 2 ratings, **Unpopular movie trimming**. To be specific, if a movie in the test set has received more than 2 ratings in the entire dataset then we delete that movie from the test set and do not predict the rating of that movie using the trained filter. The plot looks skewed indicating since there are fewer movies with less ratings in the dataset.

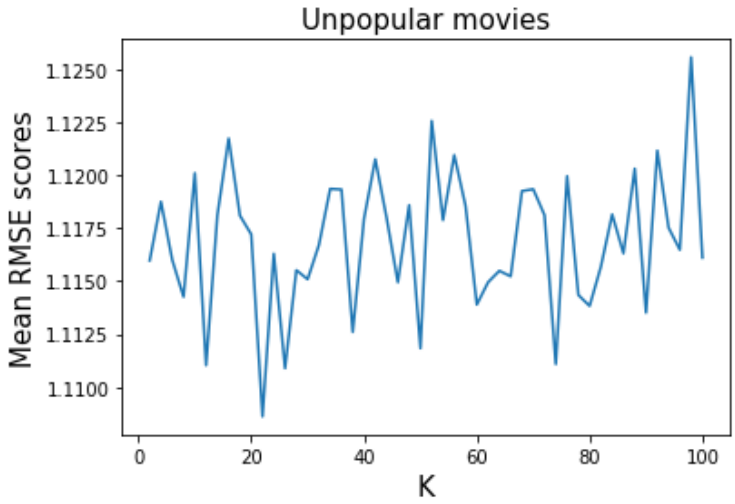

Figure 8: Plot of average RMSE against k for Popular movie trimming

Finally, we look at dataset with **High variance movie trimming**. In this trimming, we trim the test set to contain movies that have variance (of the rating values received) of at least 2 and has received at least 5 ratings in the entire dataset. To be specific, if a movie has variance less than 2 or has received less than 5 ratings in the entire dataset then we delete that movie from the test set and do not predict the rating of that movie using the trained filter.

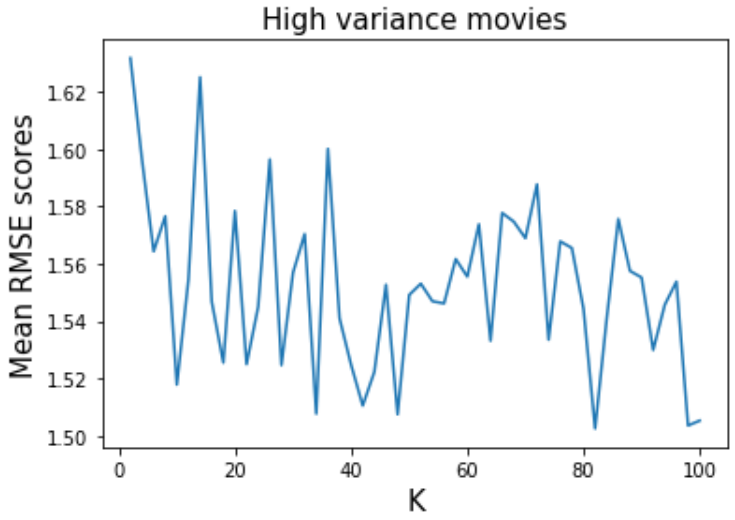

Figure 9: Plot of average RMSE against k for Popular movie trimming

The minimum average RMSE and corresponding optimal values of k are shown in the table below for each of the test sets.

Test Set	**Popular movies**	**Unpopular movies**	**High-variance movies**
Best k	48	22	82
Lowest RMSE	0.872552	1.10861	1.502594

Table 1: Optimal k-values and lowest RMSE on the trimmed test sets.

Question 15

Lastly, we look at the ROC curves for the k-NN collaborative filter designed for the whole dataset, we consider the following threshold values [2.5, 3.0, 3.5, 4.0]. We use $k = 20$ from question 11 for our ROC plots.

The area under the curve for aforementioned ROC's is as follows:

Threshold = 2.5	Threshold = 3.0	Threshold = 3.5	Threshold = 4.0
0.7874	0.7771	0.7688	0.7716

Table 2: Area under the curve for ROC with varying thresholds

We observe that the area under the curve for threshold = 2.5 is the largest, hence the natural split of ratings centers at 2.5.

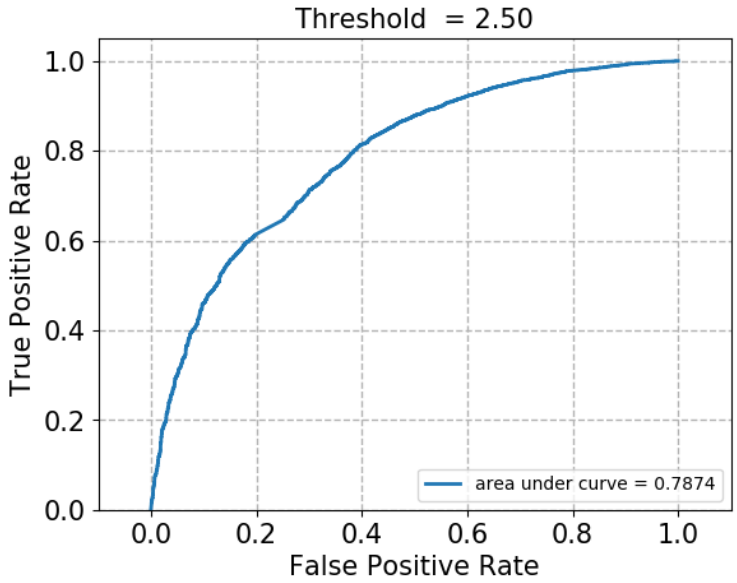

Figure 10: ROC plot for threshold of 2.5

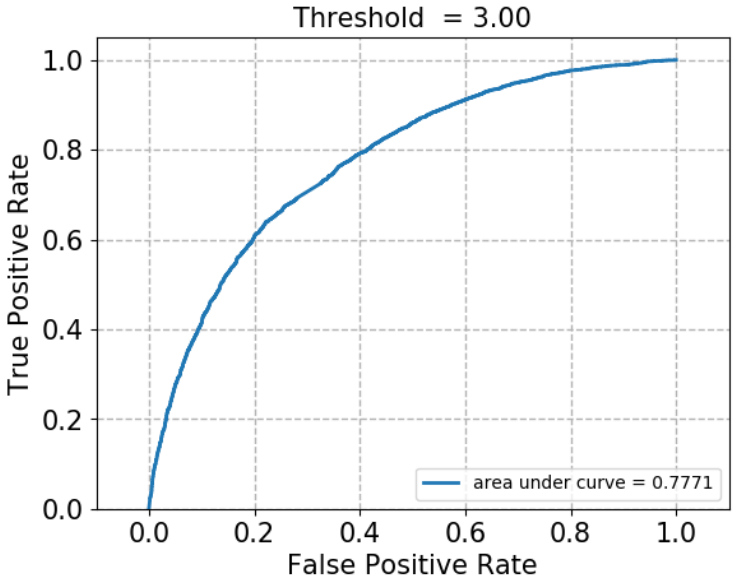

Figure 11: ROC plot for threshold of 3.0

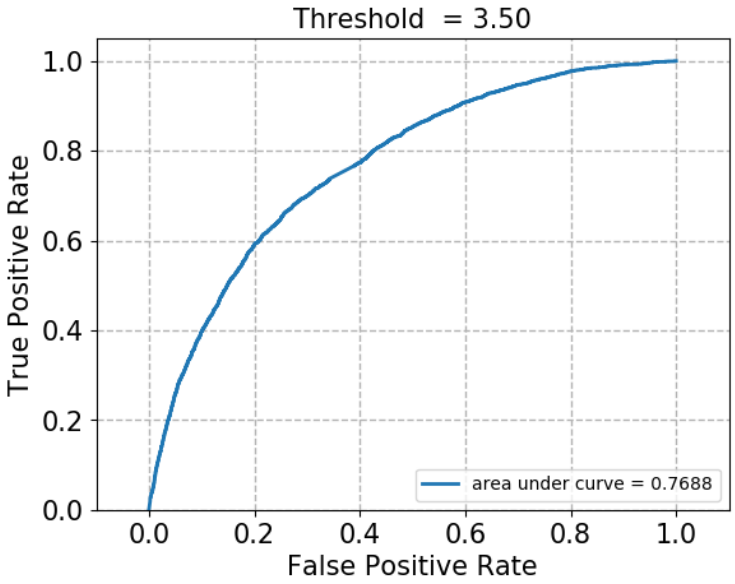

Figure 12: ROC plot for threshold of 3.5

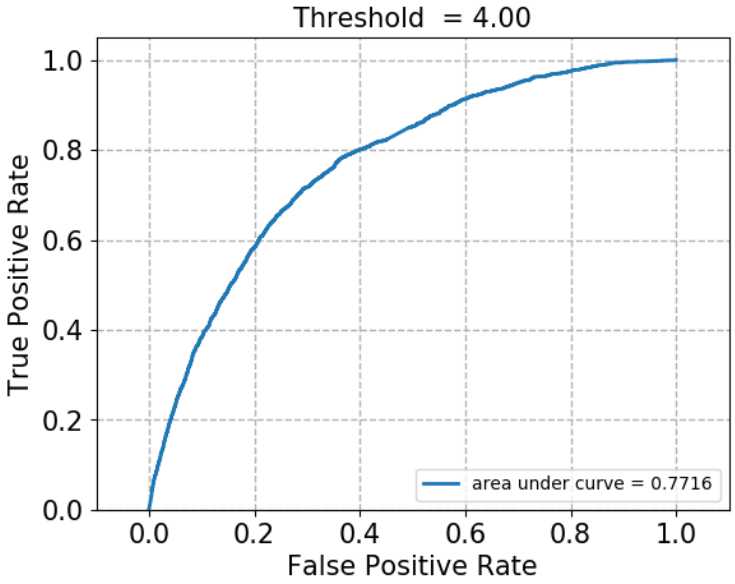

Figure 13: ROC plot for threshold of 4.0

Question 16

For questions 16-23, we implement a latent factor based model for collaborative filtering. Our latent factor based model allows us to use a direct method to fill missing entries of the rating matrix R, allowing us to predict what our users will probably like, even though have not explicitly rated those items. We do this using matrix factorization, taking advantage of the fact that the rows and columns of our rating matrix are correlated, allowing us to approximate R by a low rank matrix.

For the following sections, we choose to use non-negative matrix factorization (where U and V must not be negative). This allows us to more easily interpret our results compared to other matrix factorization methods. In addition, although in the project we are using stochastic gradient descent as our optimization algorithm, the alternating least-squares algorithm is also good as it has a faster convergence rate and is more stable.

If we define a weight matrix, W_{ij}, such that $W_{ij} = 1$ when r_{ij} is known and 0 when r_{ij} is unknown, we can formulate our optimization problem as

$$\min_{U,V} \sum_{i=1}^{m} \sum_{j=1}^{n} W_{ij}(r_{ij} - (UV^T)_{ij})^2$$

We see that this optimization problem on its own is not convex. If we consider the case where R, U, and V are scalars (dimension 1x1, r, u, v), it is clear to see that it is not convex.

$$\min_{u,v \geq 0} (r^2 - 2ruv + u^2v^2)$$

To check whether this is convex, we can find that the Hessian of this becomes $\begin{bmatrix} 2v^2 & 4uv - 2r \\ 4uv - 2r & 2u^2 \end{bmatrix}$, which is not positive semi-definite for all r, u, $v \geq 0$.

However, if we fix U, it does become convex, and we can formulate the problem as a least squares optimization problem as follows:

$$\min_{V} \sum_{i=1}^{m} \sum_{j=1}^{n} W_{ij}(r_{ij} - u_i^T v_j)^2$$

Firstly, we know this is convex because $h(x) = x^2$ is convex, and $g(v) = r_{ij} - u_i^T v$ is affine, so $h(g(v))$ will also be convex.

Lastly, W_{ij} is non-negative, and a non-negative sum of convex functions is also convex. So the objective is convex, and without constraints, the optimization problem is as well.

Question 17

In this question, we take advantage of the python surprise package's NMF function to predict the ratings of the movies in the MovieLens dataset. We then evaluate its performance using 10-fold cross-validation. We vary the number of latent factors from 2 to 50 in step sizes of 2, and compute the average RMSE and average MAE. The results of these experiments can be seen in the plots blow:

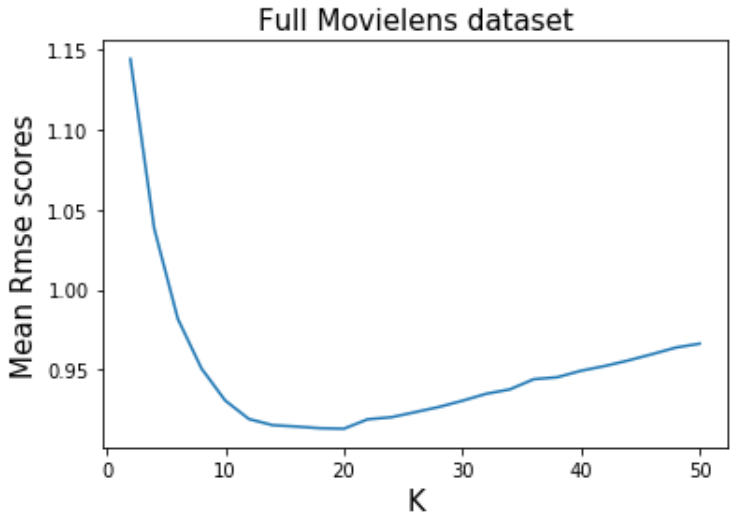

Figure 14: Mean RMSE versus number of latent factors (k) for the full Movielens dataset

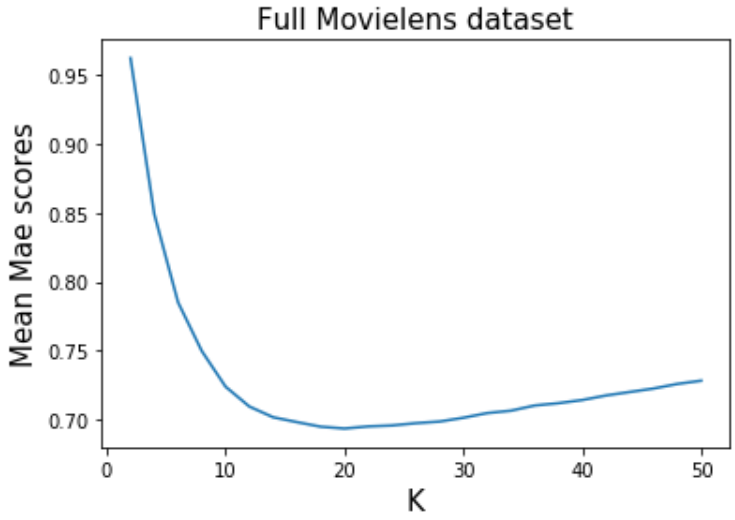

Figure 15: Mean MAE versus number of latent factors (k) for the full Movielens dataset

Question 18

We can see by observing the values in question 17 that the **optimal number of latent factors (k) is 20, as this k value gave us the lowest average RMSE and average MAE of 0.913 and 0.694** respectively. As there are 18 movie genres in the dataset, the optimal number of latent factors we found is very close to the number of movie genres.

Question 19-21

As we did previously in questions 12-14, we repeat our algorithm, NMF, on a trimmed portion of the dataset. As before, we consider the following trimmings:

1. Popular movie trimming
2. Unpopular movie trimming
3. High variance movie trimming

The minimum average RMSE for all the trims is in the table below.

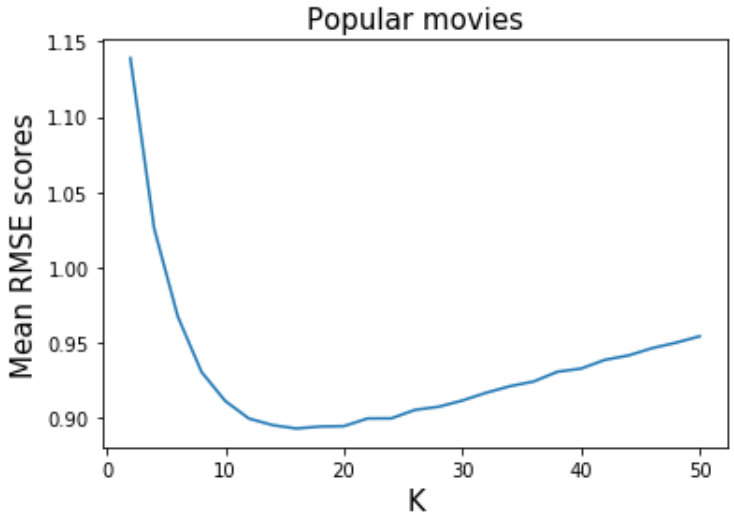

Figure 16: Mean RMSE versus number of latent factors (k) for the popular movies in Movielens dataset

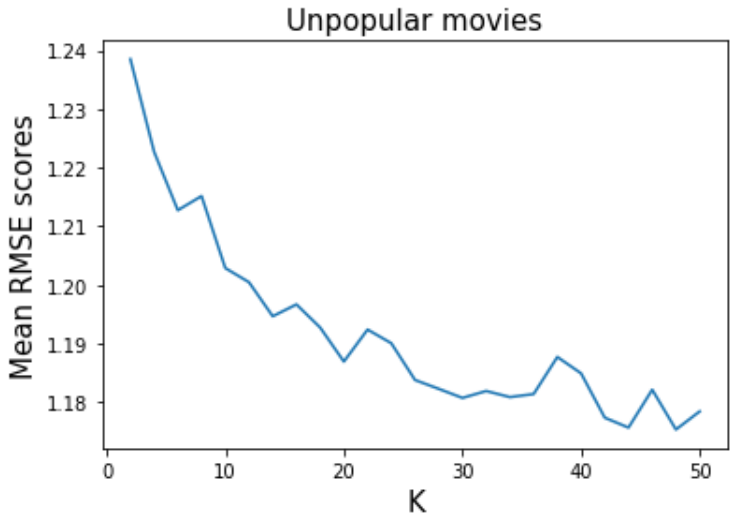

Figure 17: Mean RMSE versus number of latent factors (k) for the unpopular movies in the Movielens dataset

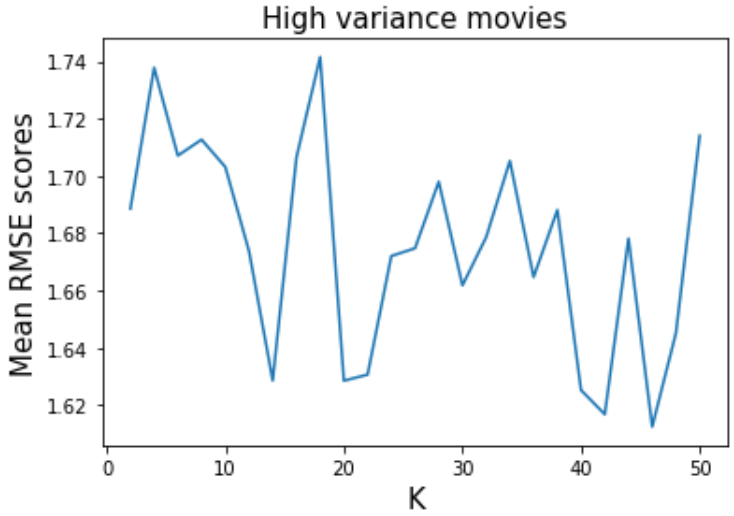

Figure 18: Mean RMSE versus number of latent factors (k) for the high variance movies in the Movielens dataset

Test Set	**Popular movies**	**Unpopular movies**	**High-variance movies**
Best k	16	48	46
Lowest RMSE	0.893999	1.178460	1.645121

Table 3: Optimal k-values and lowest RMSE for the NNMF collaborative filter on the trimmed test sets.

Question 22

Similar to question 15, we evaluate the performance of our NNMF-based collaborative filter using an ROC curve. As before, we look at the ROC curves for the whole dataset, and we consider the following threshold values [2.5, 3.0, 3.5, 4.0]. For these ROC curves, we use the optimal number of latent factors we found in question 18, $k = 20$.

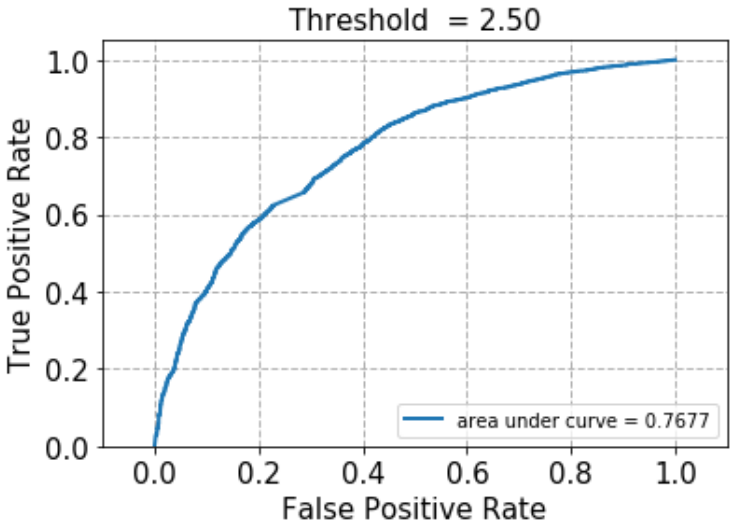

Figure 19: NMF ROC plot for threshold of 2.5

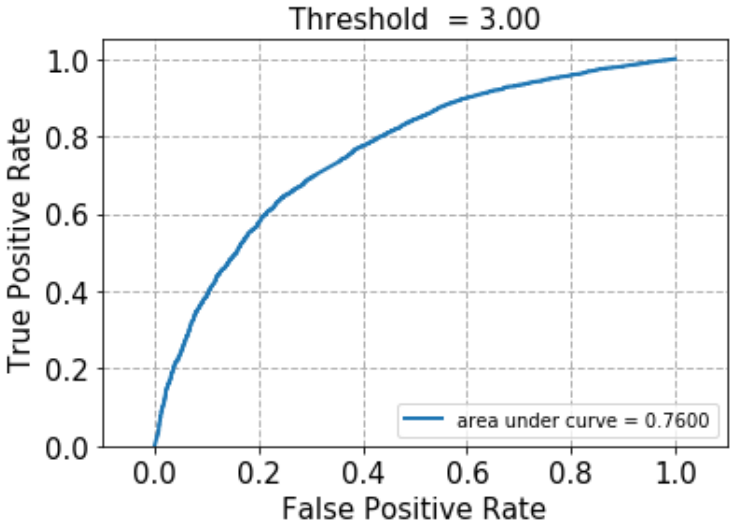

Figure 20: NMF ROC plot for threshold of 3.0

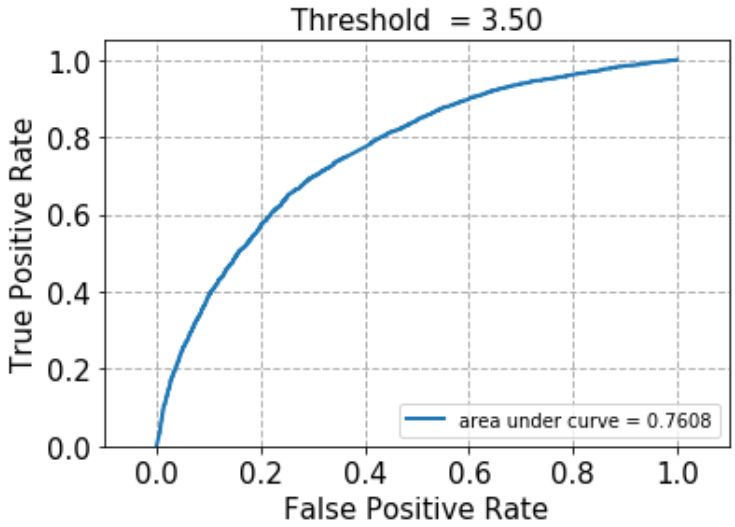

Figure 21: NMF ROC plot for threshold of 3.5

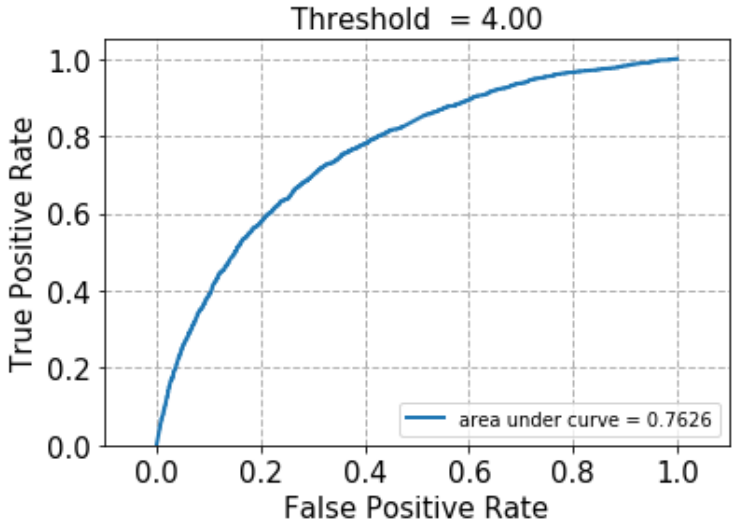

Figure 22: NMF ROC plot for threshold of 4.0

Threshold = 2.5	Threshold = 3.0	Threshold = 3.5	Threshold = 4.0
0.7677	0.7600	0.7608	0.7626

Table 4: Area under the curve for ROC with varying thresholds

Question 23

We then perform non-negative matrix factorization on the ratings matrix R to obtain the factor matrices U and V. Using V, which represents the movie-latent factors interaction (with a k of 20), we sort each column of V and find the genres of the top 10 movies for a couple of columns:

Column 1	Column 2	Column 3
Comedy, Drama, Romance	Drama	Comedy, Crime, Drama, Mystery, Thriller
Action, Drama, Sci-Fi, Thriller	Action, Comedy	Drama
Crime, Drama, Mystery, Thriller	Action, Comedy, Sci-Fi	Thriller
Comedy, Drama, Musical, Sci-Fi	Fantasy, Mystery, Romance, Thriller	Drama, Mystery, Thriller
Crime, Horror, Thriller	Action, Adventure, Fantasy	Action, Comedy, Documentary
Drama	Action, Adventure, Comedy, Fantasy	Film-Noir
Drama	Comedy, Drama, War	Comedy, Drama
Horror	Action, Drama	Drama
Drama	Action, Crime, Drama, Thriller	Comedy, Romance
Drama, Thriller	Action, Crime, Horror	Crime, Film-Noir

From the top ten movies above, we see that Comedy and Drama are the two most popular genres of movies we found. From our tests, we can see that the number of latent factors is directly proportional to the number of key groups/trends of movie genres detected in V. If we have smaller number of latent factors, we will have more genres per group and less distinct groups overall. With a smaller number of latent factors, the trends of U needs to be captured by fewer features. So, users who may have been differentiated with more columns are now grouped as one, increasing the instances of particular movie genres, in our case drama and comedy.

Additionally, we can see that each of the columns of V correspond to a small collection of genres. For example, in the table above, the first column has mostly Drama, Thriller, and Horror movies. The second column has primarily Action and Adventure movies. Finally, the last column has Crime, Drama, Mystery, and Thriller movies.

Question 24

For questions 24-29 we use matrix factorization with bias. In MF with bias, we add a bias term to the NMF cost function. Now, instead of optimizing over U and V, we optimize over U, V, b_u, and b_i.

Here, we use the surprise packages SVD algorithm to design a MF with bias collaborative filter to predict the ratings of the movies in the MovieLens dataset. We then evaluate the performance using 10-fold cross validation. We sweep the number of latent factors from 2 to 50 in step sizes of 2, then compute the average RMSE and average MAE obtained across all 10 folds. The results of this can be seen in the figures below:

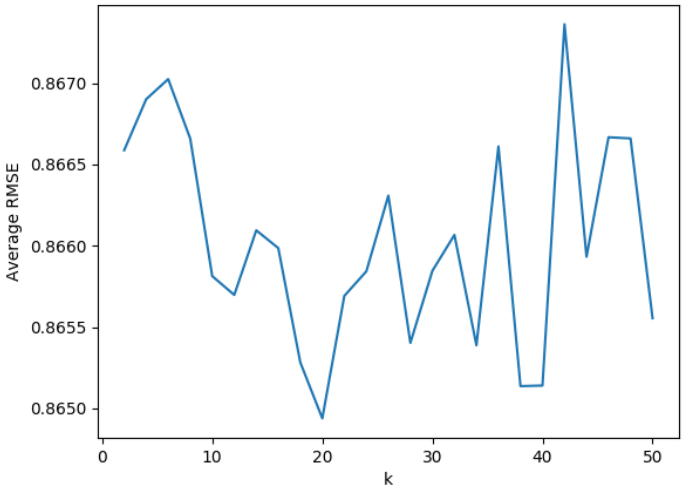

Figure 23: Mean RMSE versus number of latent factors (k) for the full Movielens dataset, using MF with bias

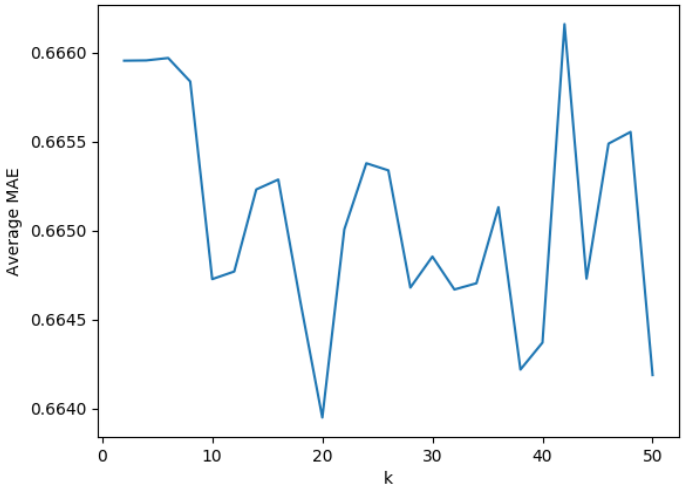

Figure 24: Mean MAE versus number of latent factors (k) for the full Movielens dataset, using MF with bias

Question 25

We then use the plots above, from question 24, to find the optimal number of latent factors, where this is the value of k that gives the minimum average RMSE or the minimum average MAE. We see that this **optimal number of latent factors is k = 20. and the lowest average RMSE and average MAE is 0.864953 and 0.663982 respectively**.

Question 26-28

As we did previously in questions 12-14, and 19-21, we repeat our algorithm, MF with bias, on a trimmed portion of the dataset. As before, we consider the following trimmings:

1. Popular movie trimming
2. Unpopular movie trimming
3. High variance movie trimming

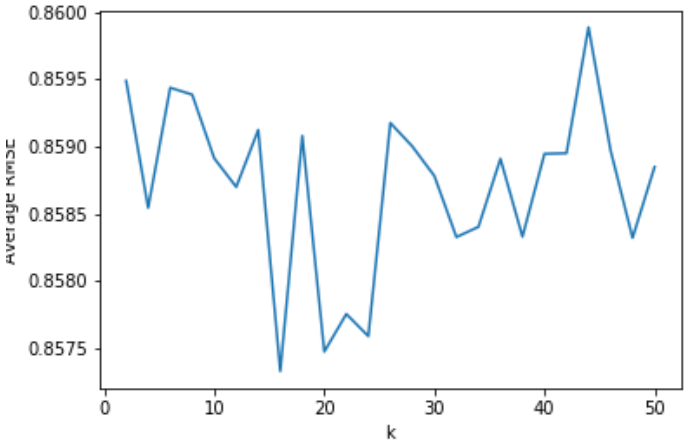

Figure 25: Mean RMSE versus number of latent factors (k) for the popular movies in Movielens dataset, MF with bias

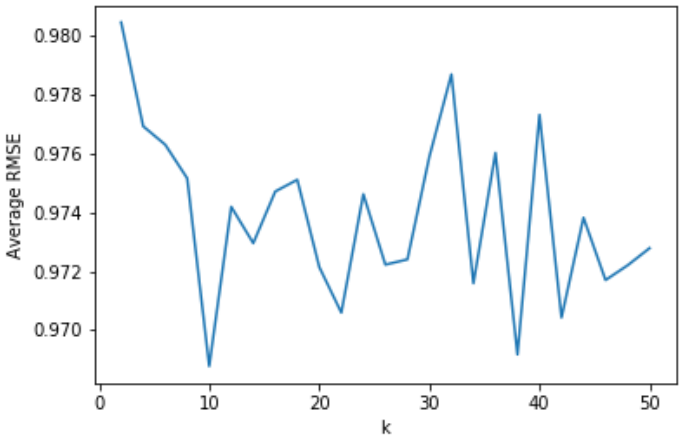

Figure 26: Mean RMSE versus number of latent factors (k) for the unpopular movies in the Movielens dataset, MF with bias

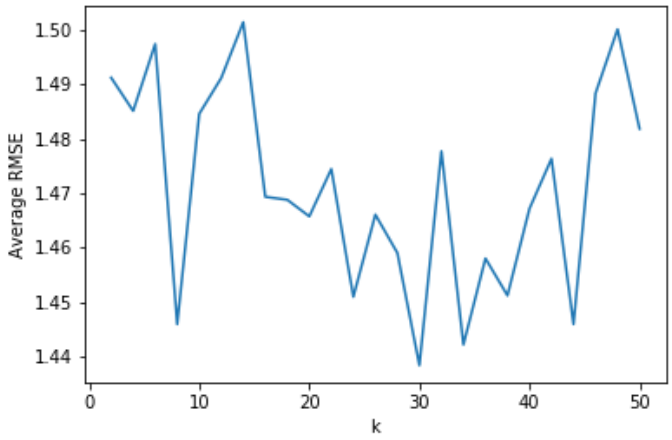

Figure 27: Mean RMSE versus number of latent factors (k) for the high variance movies in the Movielens dataset, MF with bias

From the figures above, we can find the minimum average RMSE for the popular, unnpopular, and high variance trim. The results of this can be seen in the table below.

Test Set	**Popular movies**	**Unpopular movies**	**High-variance movies**
Best k	16	10	30
Lowest RMSE	0.857848	0.968431	1.446470

Table 5: Optimal k-values and lowest RMSE for the MF collaborative filter on the trimmed test sets.

Question 29

Similar to question 15 and 22, we evaluate the performance of our MF with bias-based collaborative filter using an ROC curve. As before, we look at the ROC curves for the whole dataset, and we consider the following threshold values [2.5, 3.0, 3.5, 4.0]. For these ROC curves, we use the optimal number of latent factors we found in question 25, k = 20.

Threshold = 2.5	Threshold = 3.0	Threshold = 3.5	Threshold = 4.0
0.8052	0.7892	0.7868	0.7807

Table 6: Area under the curve for ROC with varying thresholds

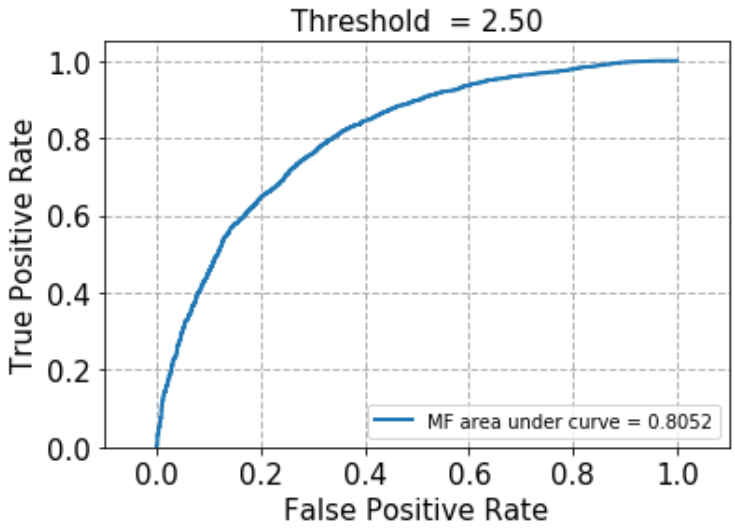

Figure 28: MF with bias ROC plot for threshold of 2.5

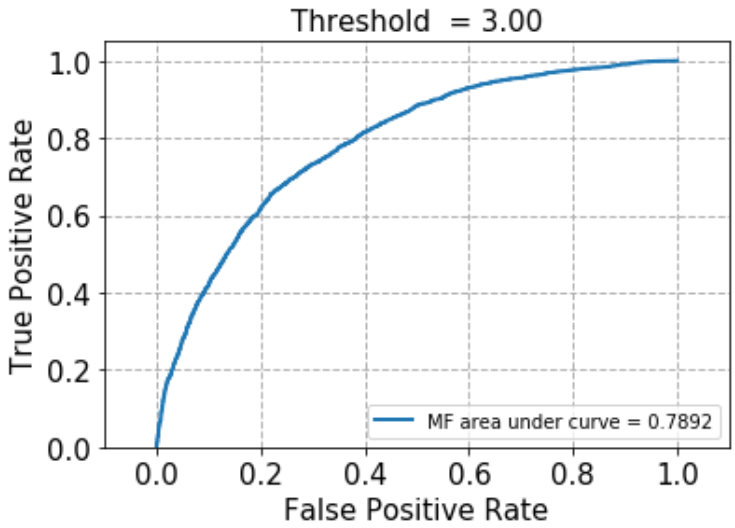

Figure 29: MF with bias ROC plot for threshold of 3.0

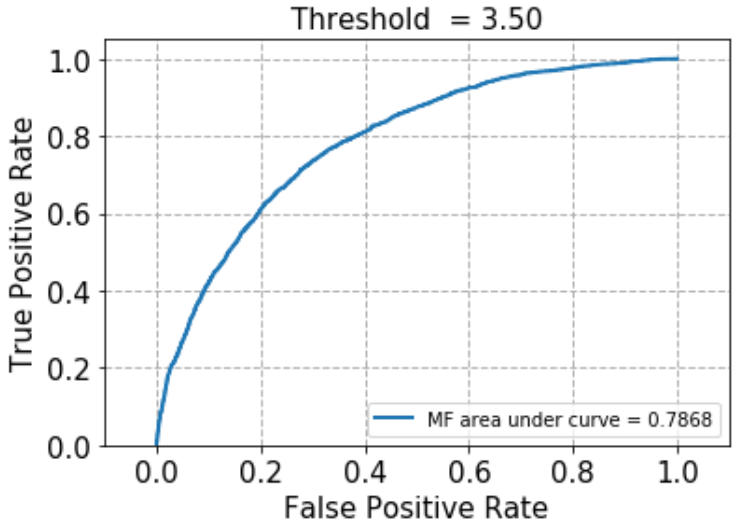

Figure 30: MF with bias ROC plot for threshold of 3.5

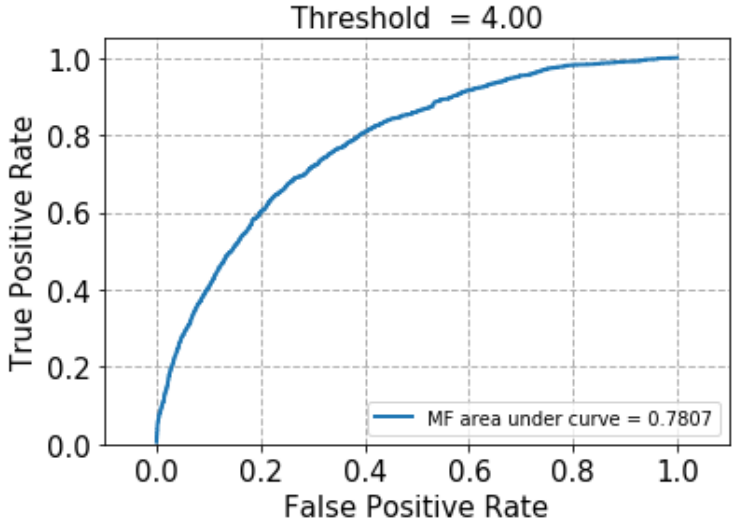

Figure 31: MF with bias ROC plot for threshold of 4.0

Question 30 - 33

Following more sophisticated filtering methods we now consider a Naive collaborative approach such that the predicted rating of user i for a movie j is given by,

$$r_{ij} = \mu_i$$

Similar to the previous sections, we use a 10-fold classification. As mentioned in the specification, we compute the mean of each user across the entire data set. There exists no training per se. The test folds have their RMSE computed with the mean values, and the average value is reported.

```
user_mean = ret_user_means(ret_user_dict(data))
kf = KFold(n_splits=10)
for trainset, testset in kf.split(data):
        test_manip = trim(data, testset, choice)
        RMSE_per = sqrt(mean_squared_error([y[2] for y in test_manip], [
    user_mean.get(x[0]) for x in test_manip]))
        RMSE = RMSE + RMSE_per
return RMSE / 10.0
```

Now we attempt the 10-fold cross validation with multiple trimming options and analyze the RMSE to derive a baseline model for collaborative filtering. The test bench is written as follows:

```
    print "****** Naive Collaboratory Filtering ******"
    print ""
    print "Method of Trim.\t\t\tRoot Mean Squared Error"
    print "No trimming\t\t\t" + str(ret_avg_rmse_10(data, 0)) #30
    print "Popular Movie Trimming\t\t" + str(ret_avg_rmse_10(data, 1)) #
    31
    print "Unpopular Movie Trimming\t" + str(ret_avg_rmse_10(data, 2)) #
    32
    print "High Var Movie Trimming\t\t" + str(ret_avg_rmse_10(data, 3))
    #33
    print ""
```

Now we obtain the RMSE for all 4 approaches. The output of our code is generated and printed below in the table:

Method of trimming	Root Mean Squared Error
No Trimming	0.934707
Popular Movie Trimming	0.932313
Unpopular Movie Trimming	0.970655
High Variance Movie Trimming	1.453530

Table 7: Naive Collaboratory Filtering

Question 34

Now that we have explored the three models for recommendation systems, we do a comparative study between the 3 models to find which one performs the best. Find the ROC plot comparison below for threshold = 3.

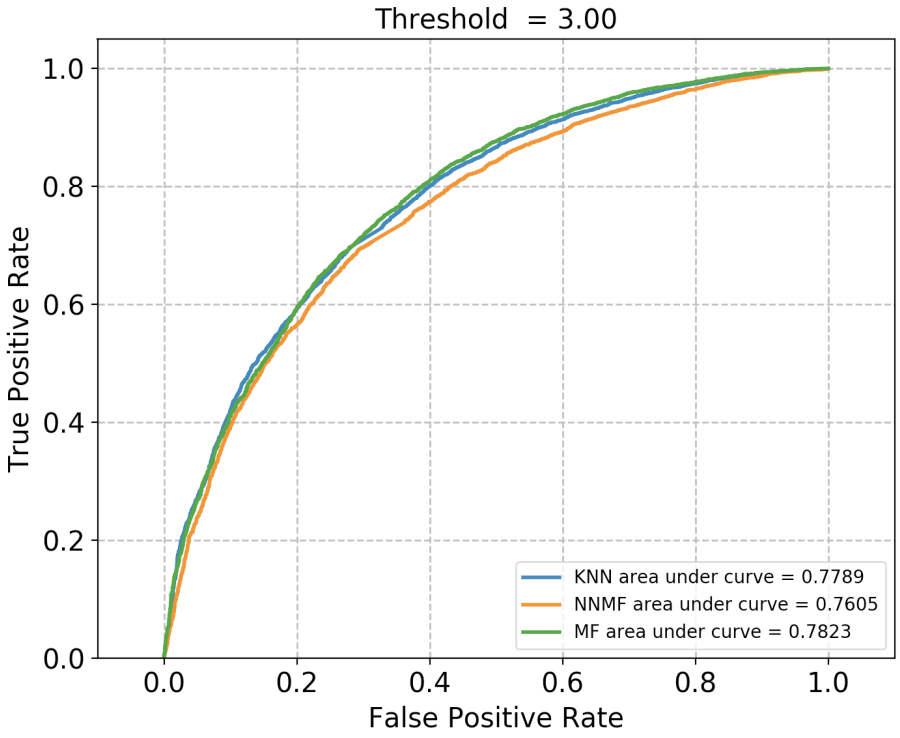

Figure 32: Comparative ROC plot - KNN, NNMF, MF (Threshold = 3)

It is clear from this graph that the **MF model performs best with KNN a close second**. It is also important to recognize that while the KNN and MF models at threshold = 3 are relatively equal in performance, **other thresholds reveal a wider disparity between the two models, making the MF model the clear winner.**

Question 35

Precision describes the **rate at which a suggested recommendation is liked by the user**. The recall is simply **the fraction of recommended movies that show up in user's likes**. Both of these measures are useful in recommendation systems; it is beneficial to have a large fraction of items from a user's favorites, and have a large fraction of recommendations that a user prefers.

Question 36-38

The following graphs represent the precision curve, recall curve and precision-recall curve for the three methods used. Note that the metrics calculations were computed using the following code:

```
    for userid, user_ratings in test_user_dict.items():
        user_ratings = sorted(user_ratings,key=itemgetter(1))
        user_ratings.reverse()
        if len(user_ratings) < t: # Constraint 2
            continue
        # Size of G per user
        magG = sum((true_rating >= threshold) for true_rating,_,_ in
    user_ratings)
        if magG == 0: # Constraint 3
            continue
        # constraint 1
        magS = sum(bolo == False for true_rating,est_rating, bolo in
    user_ratings[:t])
        if magS == 0:
            continue
        magG_I_S = sum(((true_rating >= threshold) and (est_rating >=
    threshold) and bolo == False) for (true_rating, est_rating, bolo) in
     user_ratings[:t])
        prec = prec + magG_I_S / float(magS)
        reca = reca + magG_I_S / float(magG)
        number_of_users = number_of_users + 1.0
```

Note that the code contains the 3 constraints commented as constraints. Constraint 1 drops those predictions in S that have no ground truth, 2 only considers users with at least t ratings, and 3 looks at the number of movies liked by user.

The 3 curves for KNN, NNMF and MF are given below. Note that the precision goes down as the number of suggestions go up. This makes sense - **the intersection of the recommended set with liked movies saturates to the number of recommended movies. In contrast, the recall increases with t. A higher number of recommendations result in larger number of liked movies being hit; the ratio w.r.t liked movies goes up.**

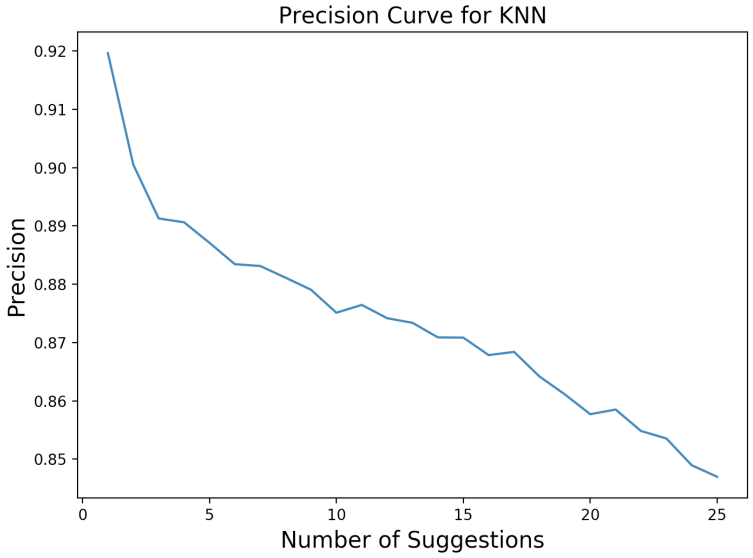

Figure 33: The precision curve is shown to have an inverse relationship with number of suggestions

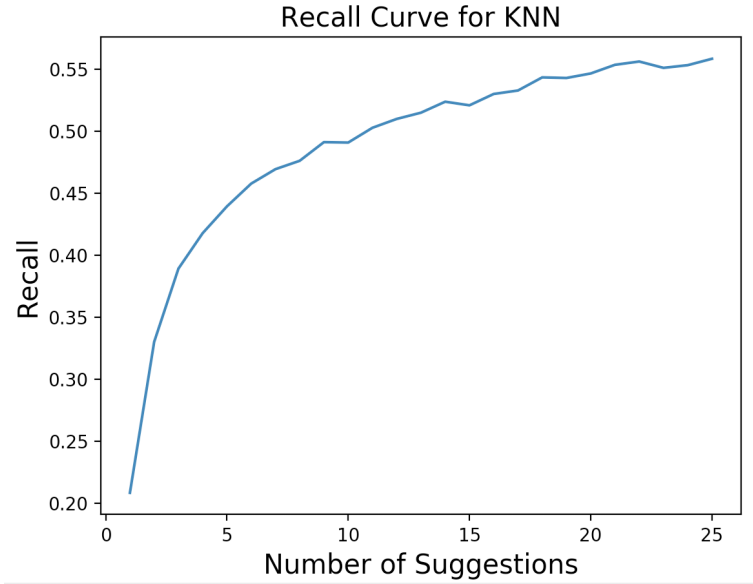

Figure 34: The recall curve for KNN shows the counter effect to the previous plot, validating our inverse relation hypothesis

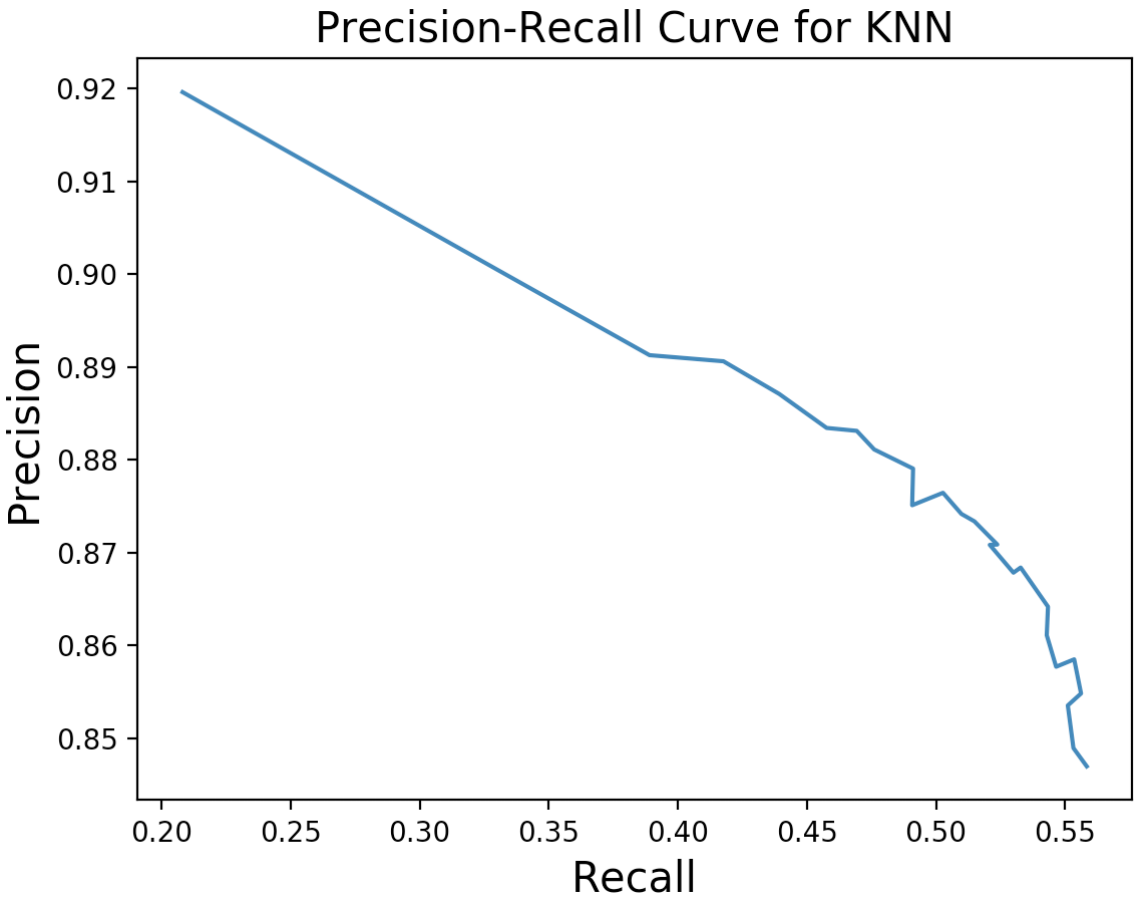

Figure 35: The precision-recall curve demonstrates the trade-off as predicted

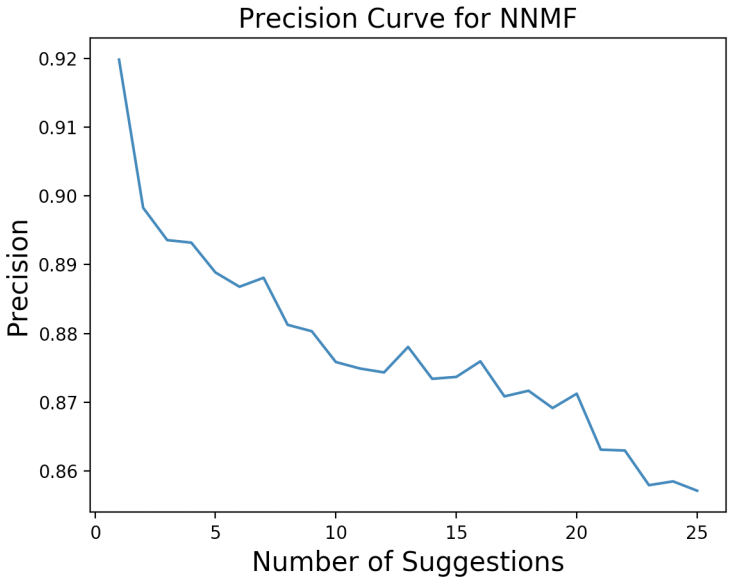

Figure 36: The precision curve for NNMF follows the trend of KNN

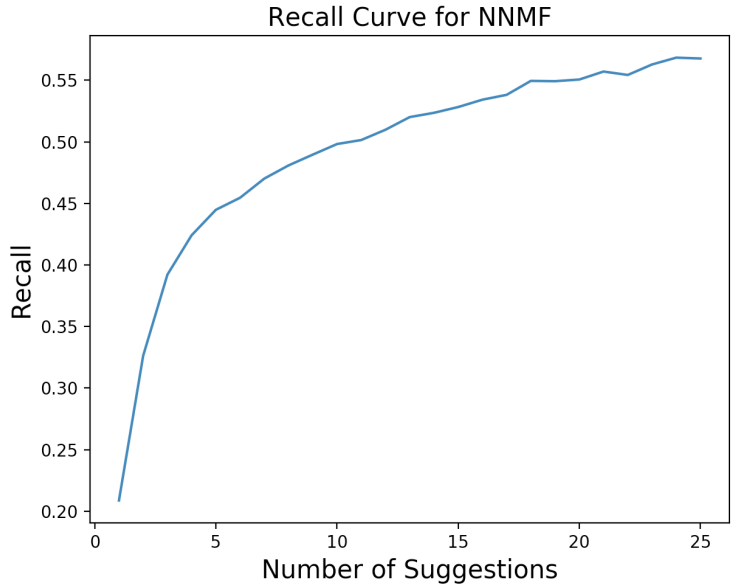

Figure 37: The Recall function for NNMF has an inverse relation to precision in the previous plot

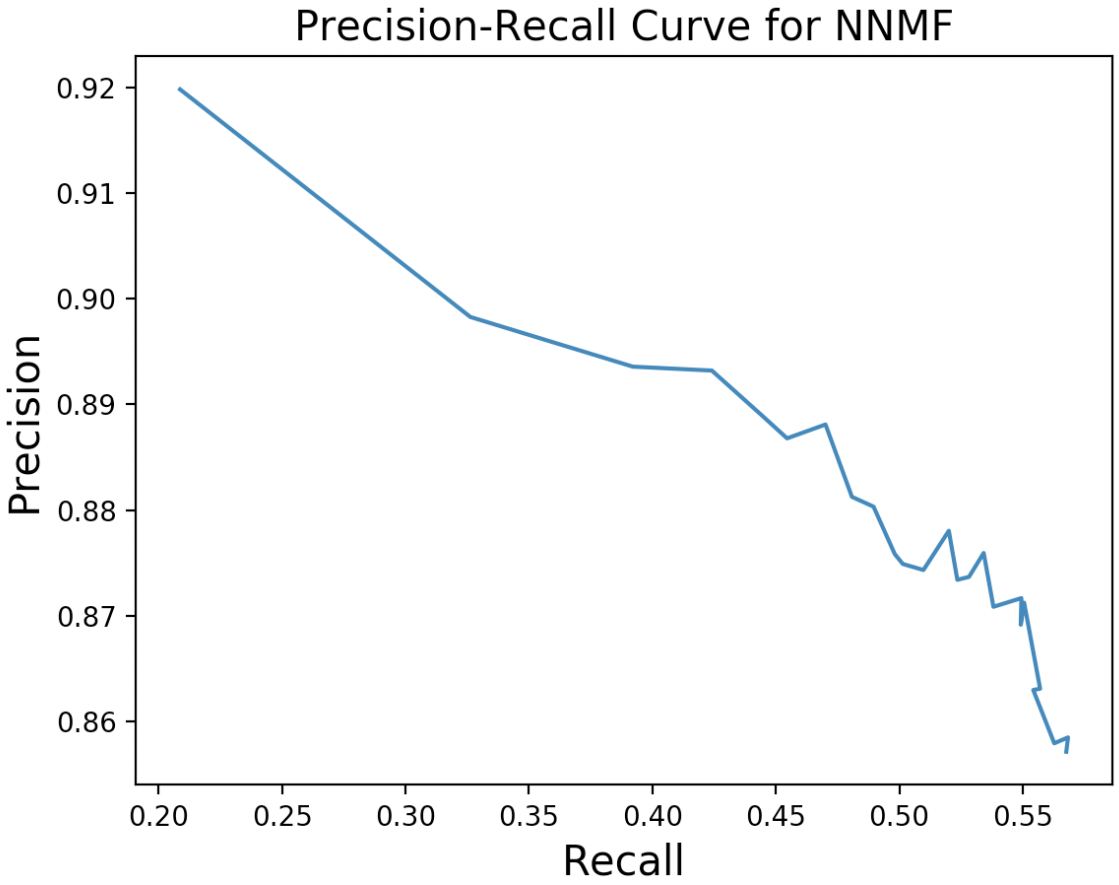

Figure 38: The trade-off curve shows expected results with precision and recall on opposite sides of the optimization

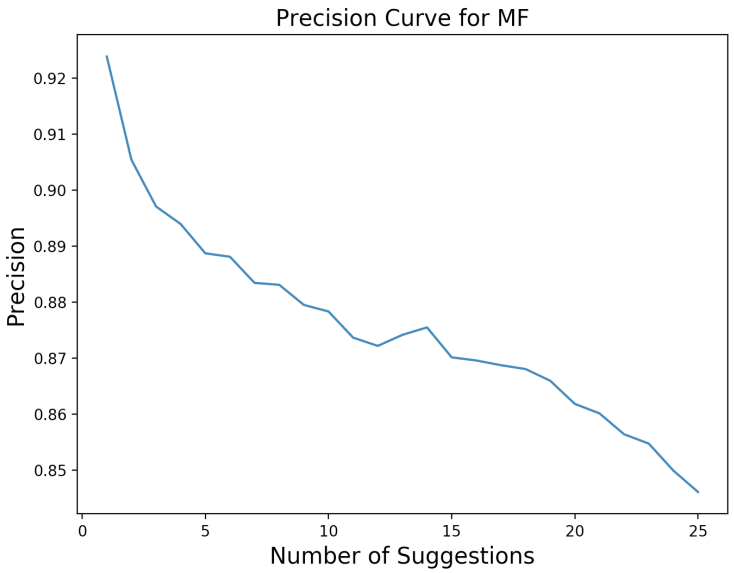

Figure 39: The same procedure as previously discussed in KNN, NNMF is used for MF

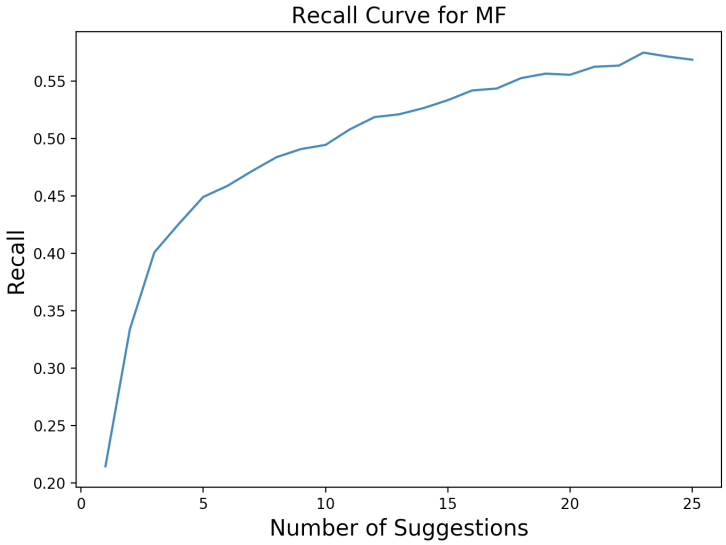

Figure 40: The trend is consistent in the recall curve

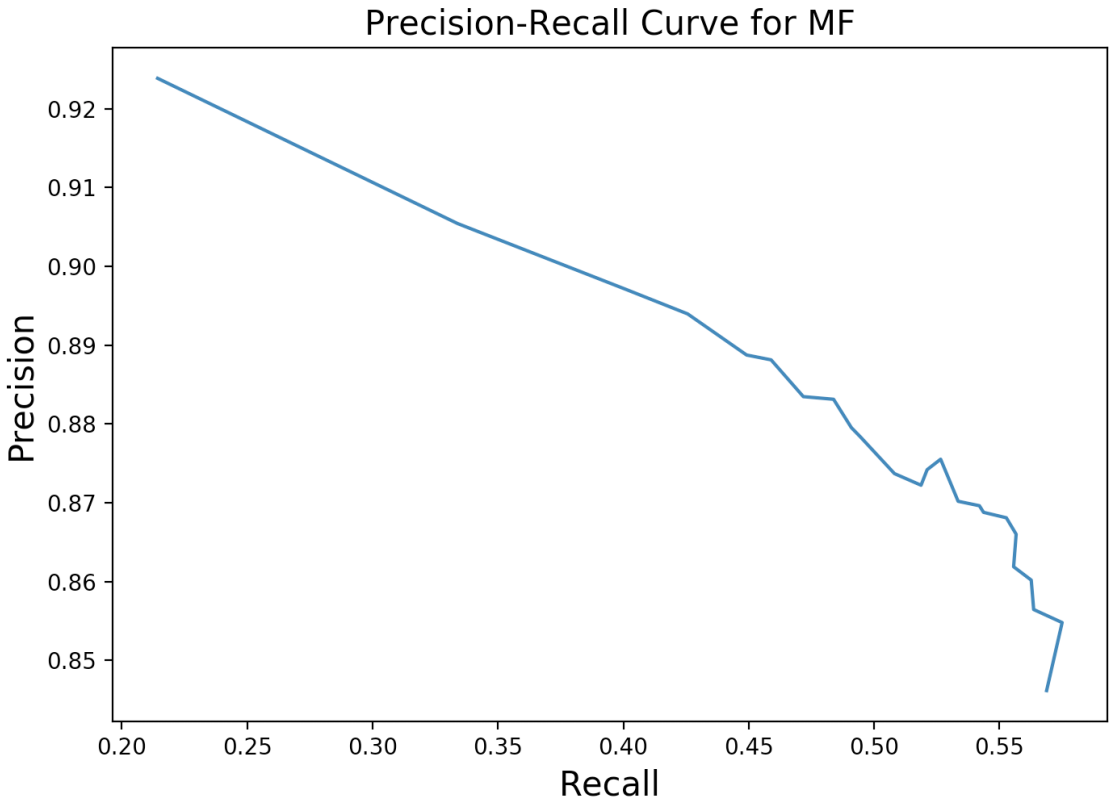

Figure 41: The precision-recall curve is also as expected

Question 39

The comparative precision-recall curves are given below. Ideally, we would like to find the curve with the most precision and recall across any number of recommendations. The graph below shows that all the models are **roughly equal** in terms of the performance. **Further inferences depends on the model's usage and the optimal operating point among the pareto-optimal options**. For lower recall requirements, MF is the clear winner. For higher recall and lower precision all models perform similarly.