ECE 219: Large Scale Data Mining: Models and Algorithms Project 3: Collaborative Filtering

Sarah Madsen, Connor Roberts, Edwin Calderon

19 February 2021

1 Introduction

Throughout Project Three, the group explores collaborative filtering through neighborhood-based collaborative filtering and model-based collaborative filtering. In particular, User-based models are assessed alongside latent factor implementations. Performance evaluations are implemented using the receiver operating characteristic curve per algorithm implemented. A performance comparison is also established using the various collaborative filters. Insights are also collected from precision and recall plots.

2 MovieLens Dataset

2.1 Question 1

A sparsity of 0.016999683055613623 was calculated.

2.2 Question 2

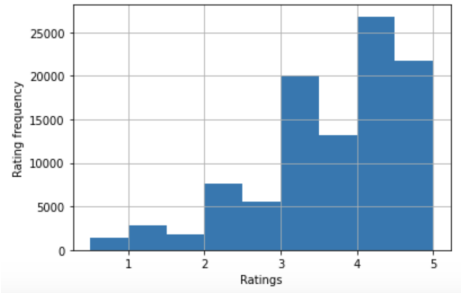


Figure 1: Histogram of the ratings of the dataset

In figure one, it is seen that the prominent ratings for most of the movies is high. The range is from 0.5 to 5.0 with a large majority given a rating of 3.0, 4.0, and 5.0.

2.3 Question 3

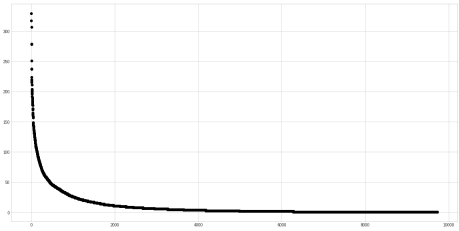


Figure 2: Distribution of ratings among movies

2.4 Question 4

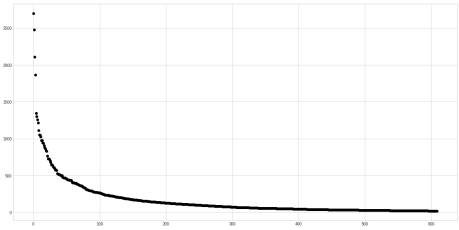


Figure 3: Distribution of ratings among users

2.5 Question 5

As we can see in the graph shown in Figure 2, there are very few movies with a large amount of ratings. Most movies in the received a small number of ratings while we see a few outliers at the very top with up to 10x the amount of ratings seen on average. This will make it difficult to accurately recommend movies with less ratings as we don't have nearly as much data as the ones at the higher end of the spectrum.

2.6 Question 6

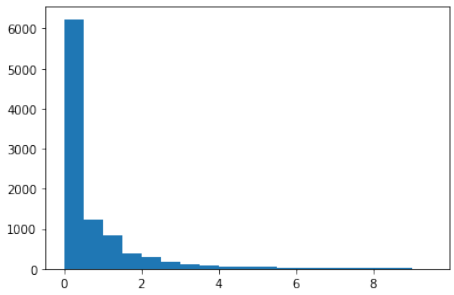


Figure 4: Histogram of variance values among movies

We can see in the figure above that most of the movies have a very low variance with the 0 to 0.5 bin being the most populous. After approximately 2, the amount of movies with high variance is very very low. This means we can expect most movies to have a consensus on what they were rated by different users and we can more accurately predict these ratings.

3 Neighborhood-based collaborative filtering

3.1 Question 7

The formula for μ_u - the mean rating for user u computed using her specified ratings - in terms of I_u and r_{uk} is the sum of the user's ratings divided by the set of item indices for which ratings have been specified by user u.

$$\mu_u = \frac{\sum_{k=1}^{N} r_{uk}}{I_u}$$

3.2 Question 8

$I_u \cap \mathrm{I}_v$ is the intersection of item indices for which there exists ratings by both users u and v. $I_u \cap \mathrm{I}_v$ can equal to zero since items may not be rated by both users.

3.3 Question 9

The intuition behind mean centering the raw ratings

$$(r_{vj} - \mu_u)$$

in the prediction function is to "normalize" the ratings with respect to a user's bias/habits. A user may typically give either high or low ratings consistently such that its weight

appears to show a high or low preference, but, with regards to other user's bias/habit, the rating may only indicates a user's average rating.

3.4 Question 10

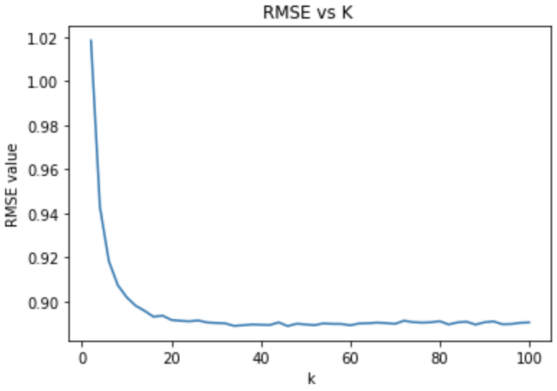


Figure 5: RMSE for knn vs. k

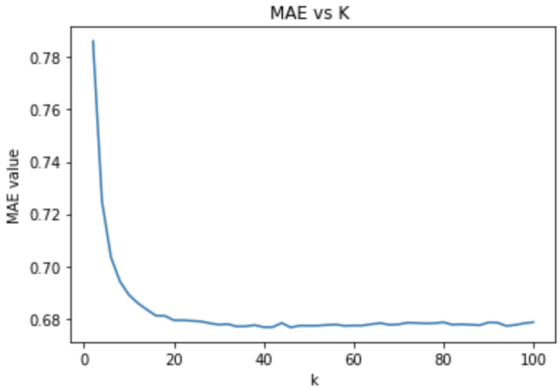
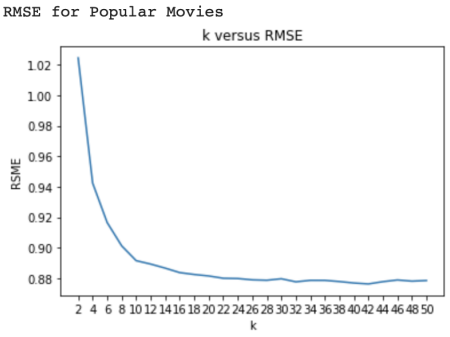


Figure 6: MAE for knn vs. k

3.5 Question 11

The 'steady-state' values for our k (14) are noted below.
Minimum k: 14
RMSE at minimum k: 0.8892693748711421
MAE at minimum k: 0.6773293379151676

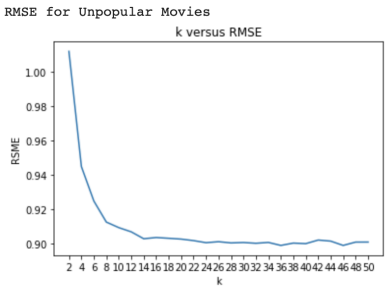
3.6 Question 12




```
Minimum RMSE value:  0.8763318701621925
at k:  42
Minimum avg RMSE:  0.8763318701621925
```

Figure 7: k-NN collaborative filter for the popular movie trimmed set

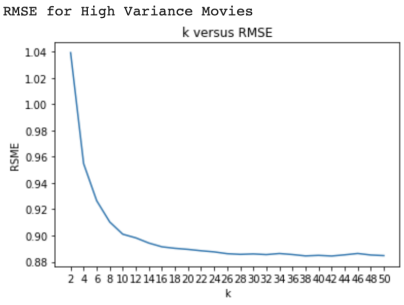
3.7 Question 13




```
Minimum RMSE value:  0.8988976004135416
at k:  36
Minimum avg RMSE:  0.8988976004135416
```

Figure 8: k-NN collaborative filter for the unpopular movie trimmed set

3.8 Question 14

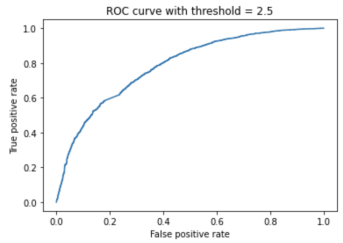



```
Minimum RMSE value:  0.8844380531227877
at k:  42
Minimum avg RMSE:  0.8844380531227877
```

Figure 9: k-NN collaborative filter for the high variance movie trimmed set

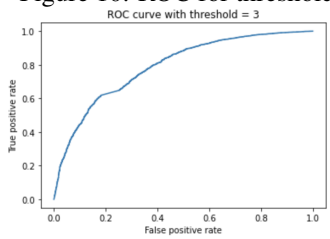
3.9 Question 15

Below in the following figures are the ROC curves including the AUC using our steady state k as the threshold changes over [2.5 -4.0] once the observed ratings are changed to a binary scale.



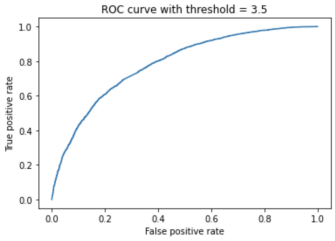

```
The area under the curve (AUC) value is 0.783143272653317
```

Figure 10: ROC for threshold = 2.5



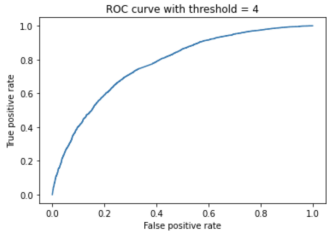

```
The area under the curve (AUC) value is 0.7908626570017433
```

Figure 11: ROC for threshold = 3.0




```
The area under the curve (AUC) value is 0.7838432232467198
```

Figure 12: ROC for threshold = 3.5




```
The area under the curve (AUC) value is 0.7749395905807956
```

Figure 13: ROC for threshold = 4

4 Model-based collaborative filtering

4.1 Question 16

The optimization problem given in equation 5 is not a convex as the Hessian for our equation below is not positive semidefinite.

$$minimize_V \|W(r - UV^T)\|^2$$

4.2 Question 17

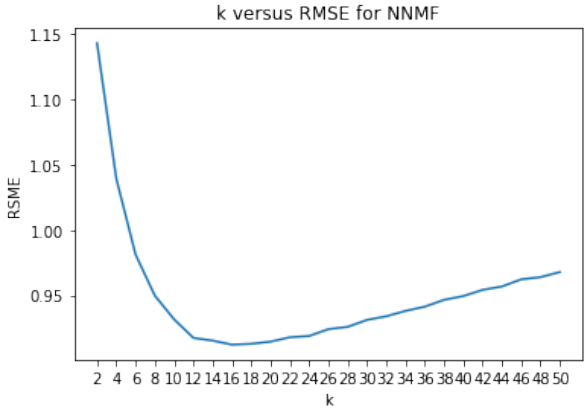


Figure 14: RMSE for NNMF vs. k

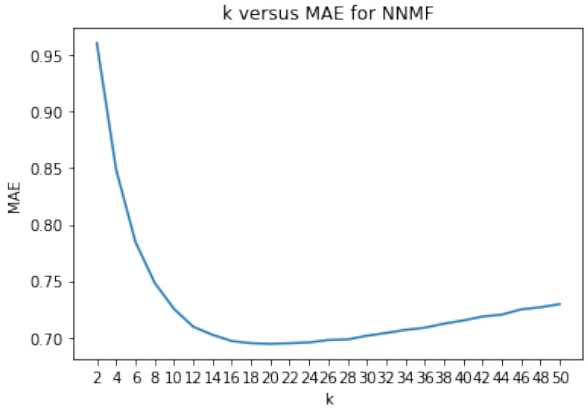


Figure 15: MAE for NNMF vs. k

4.3 Question 18

The 'steady-state' values for our k (20) are noted below.
Minimum k: 20
RMSE at minimum k: 0.9123938286631338
MAE at minimum k: 0.694289614514209
In the movielens dataset there are 18 genres that can be assigned to movies as well as having no genres listed so our minimum k of 20 was close but not exactly the number of genres in the dataset.

4.4 Question 19

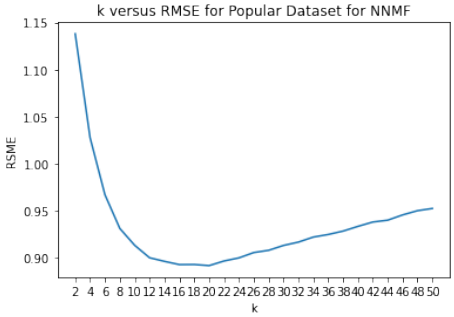


Figure 16: NNMF collaborative filter for the popular movie trimmed set

Minimum avg RMSE: 0.8913882930927883

4.5 Question 20

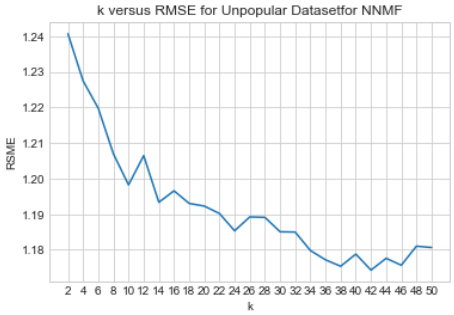


Figure 17: NNMF collaborative filter for the unpopular movie trimmed set

Minimum avg RMSE: 1.1742936332397462

4.6 Question 21

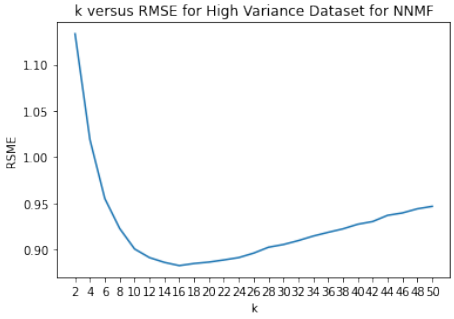


Figure 18: NNMF collaborative filter for the high variance trimmed set

Minimum avg RMSE: 0.8827169869620324

4.7 Question 22

Below in the following figures are the ROC curves including the AUC using our steady state k as the threshold changes over [2.5 -4.0] once the observed ratings are changed to a binary scale.

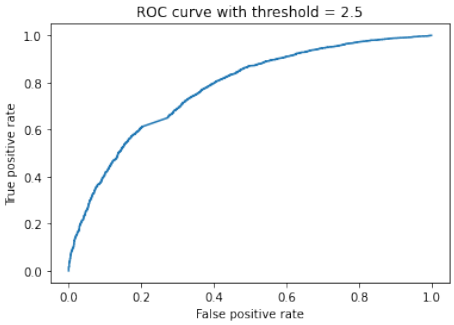


Figure 19: ROC for threshold = 2.5

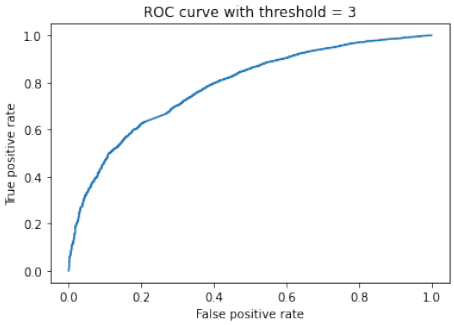


Figure 20: ROC for threshold = 3.0

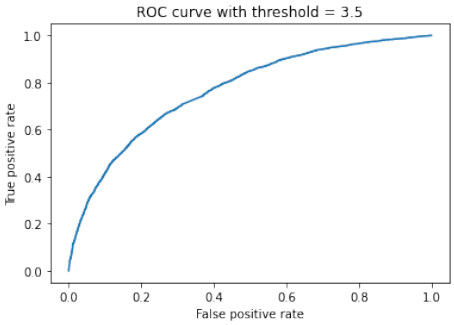


Figure 21: ROC for threshold = 3.5

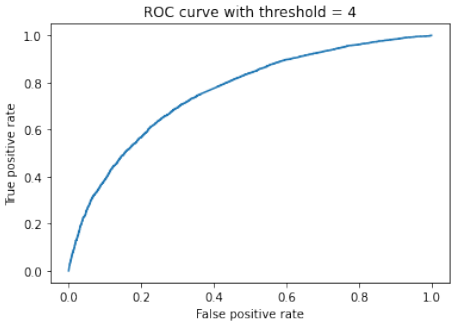


Figure 22: ROC for threshold = 4

4.8 Question 23

After looking at the genres of the top 10 movies in columns 1-3, we see 2 genres that appear multiple times in all three columns. Comedy and drama seem to the two genres contributing to the movies at the top of the list and there does appear to be a connection between this genres and the latent factors

4.9 Matrix factorization with bias

4.9.1 Question 24

See Figures 23 and 24

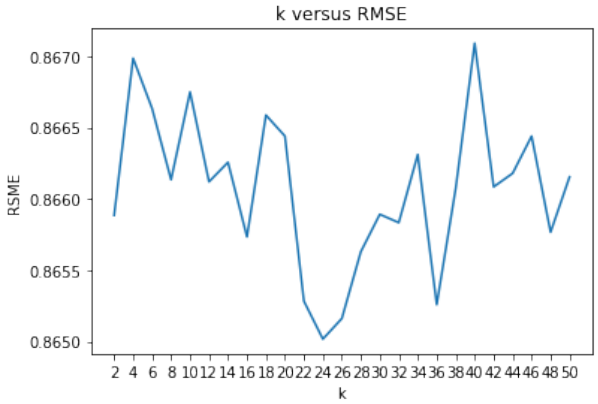


Figure 23: Average RMSE plotted against k for MF with bias.

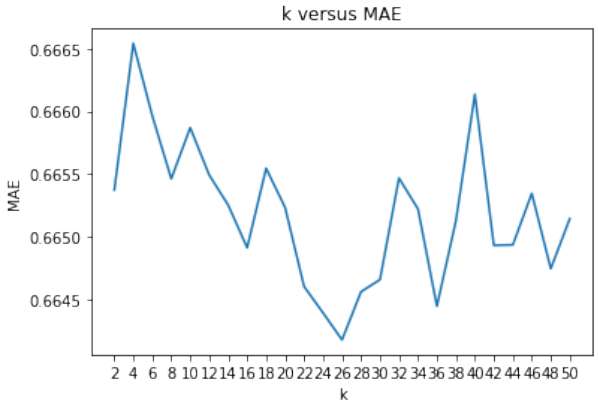


Figure 24: Average MAE plotted against k for MF with bias.

4.9.2 Question 25

For matrix factorization (MF) with bias tested on the entire data set, the minimum average RMSE is 0.86508 and the minimum average MAE is 0.66418. The optimal k to minimize RMSE is $k = 24$.

4.9.3 Question 26

For MF with bias tested on the popular movie data set, the minimum average RMSE is 0.86639 (Figure 25).

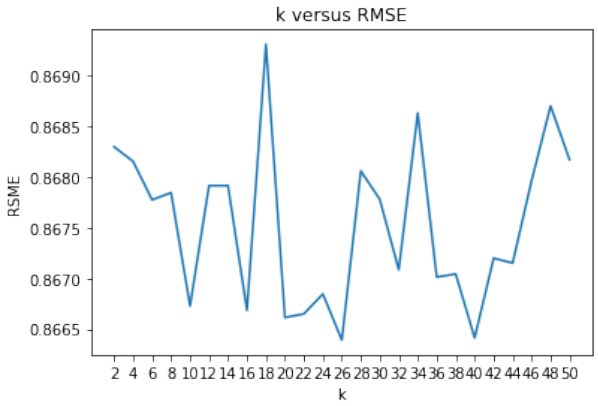


Figure 25: Average RMSE plotted against k for the popular movie data set

4.9.4 Question 27

For MF with bias tested on the unpopular movie data set, the minimum average RMSE is 0.86398 (Figure 26).

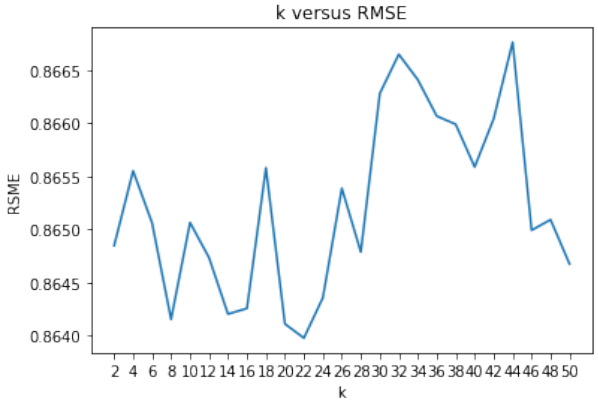


Figure 26: Average RMSE plotted against k for the unpopular movie data set

4.9.5 Question 28

For MF with bias tested on the high variance movie data set, the minimum average RMSE is 0.87176 (Figure 27).

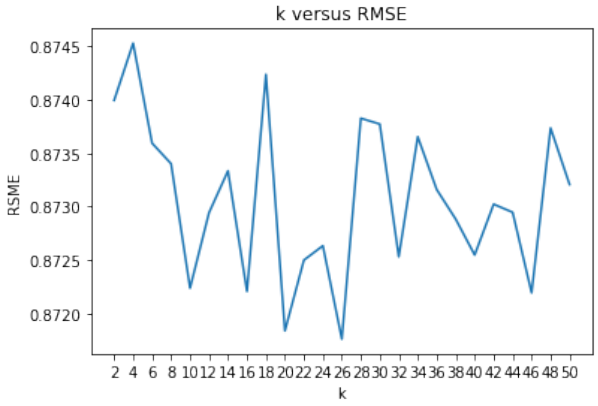


Figure 27: Average RMSE plotted against k for the high variance movie data set

4.9.6 Question 29

See Figure 28 for ROC plots and Table 1 for AUC values.

threshold	AUC
2.5	0.59495
3	0.65646
3.5	0.71241
4	0.67601

Table 1: The AUC values for various threshold levels for MF w/ bias.

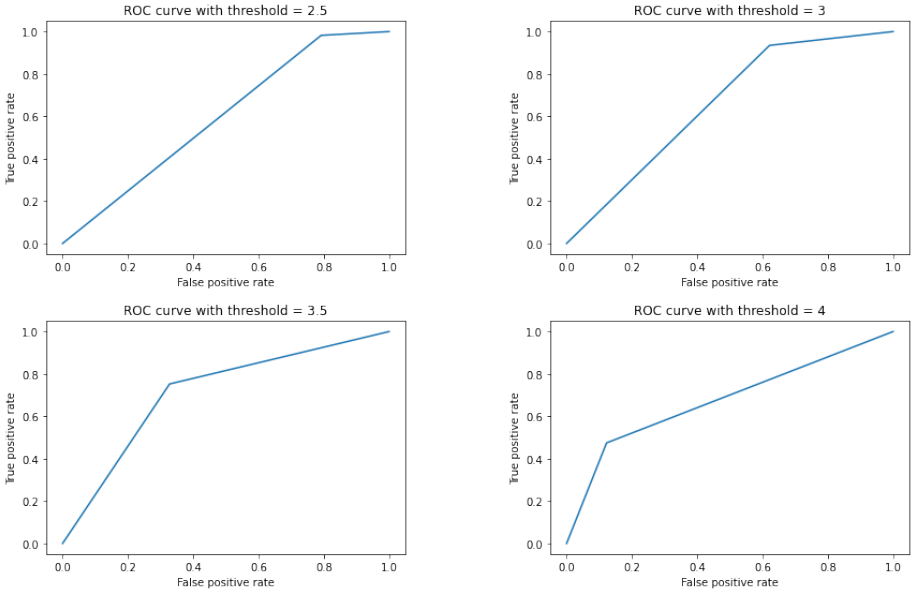


Figure 28: ROC curves for MF with bias and different binarization thresholds: 2.5 (top,left), 3 (top,right), 3.5 (bottom,left), and 4 (bottom,right)

5 Naive collaborative filtering

5.0.1 Question 30

The average RMSE across all 10 folds for the naive collaborative filter tested on the entire data set is RMSE = 0.95233.

5.0.2 Question 31

The average RMSE for the naive collaborative filter tested on the popular data set is RMSE = 0.9615.

5.0.3 Question 32

The average RMSE for the naive collaborative filter tested on the unpopular data set is RMSE = 0.94263.

5.0.4 Question 33

The average RMSE for the naive collaborative filter tested on the high variance data set is RMSE = 0.97040.

6 Performance comparison

6.1 Question 34

From Figure 29 we see that the three methods perform similarly. The worst performing algorithm is NNMF, as it has the least area-under-the-curve (AUC) and the highest false positive rate (FPR). MF with bias has a better FPR than NNFM, but performs more poorly than KNN. Since KNN has the lowest FPR, it is the best filter for predicting the ratings of movies under these conditions.

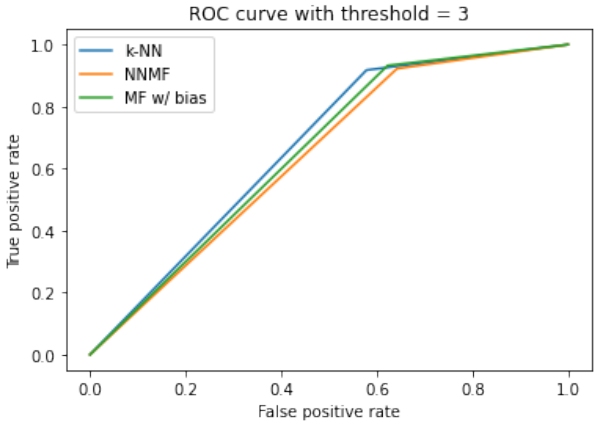


Figure 29: ROC curve for k-NN, NNMF, and MF filters at threshold = 3

7 Ranking

7.1 Question 35

In this set of equations $S(t) \cap G$ can be thought as the true positives. It is the set of movies that was both recommended for the user and that the user liked. $S(t)$ can be thought of as the total predicted positives (recommended movies), where the predicted positive means the user is predicted to like it (regardless of the whether or not the user actually liked it). G can be thought of as actual positives, meaning the user actually liked it (regardless of whether or not it was recommended). Thus, we can reformulate these equations:

$$Precision(t) = \frac{movie\ was\ recommended\ AND\ user\ liked\ it}{movie\ was\ recommended}$$

$$Recall(t) = \frac{movie\ was\ recommended\ AND\ user\ liked\ it}{movies\ user\ likes}$$

Alternatively, we can write this mathematically,

$$Precision(t) = \frac{true\ positive}{true\ positive + false\ positive}$$

$$Recall(t) = \frac{true\ positive}{true\ positive + false\ negative}$$

7.2 Question 36

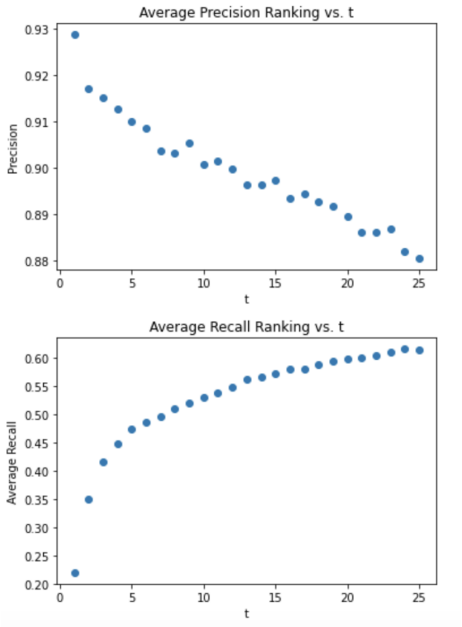


Figure 30: Average Precision and Recall ranking vs t respectively

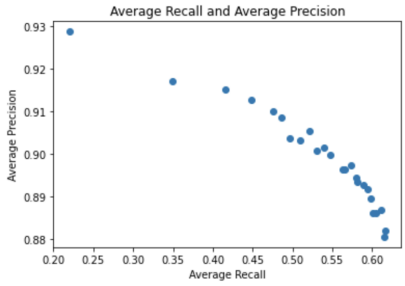


Figure 31: Average Precision ranking vs Average Recall ranking

As t sweeps, we see an inverse relationship between the Average Precision ranking and Average Recall ranking. The Average Recall figure shows a saturation as t increases, and, as expected, the Average Precision ranking inversely reaches a steady state as t increases. As the number of items recommended to the user varies, the non-

monotonic behavior of the precision plot is seen in both its plot and its plotted relation as a function of recall.

7.3 Question 37

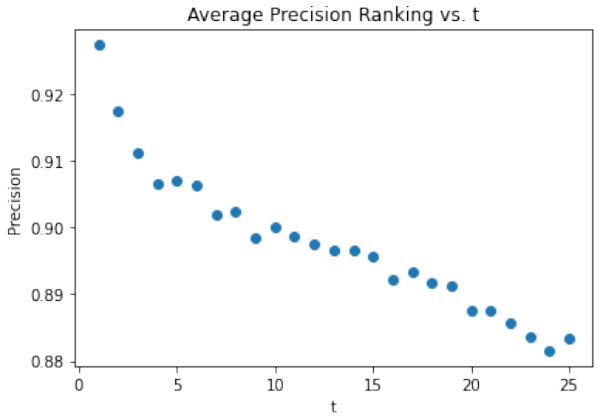


Figure 32: Precision versus t for ranking using NNMF collaborative filter.

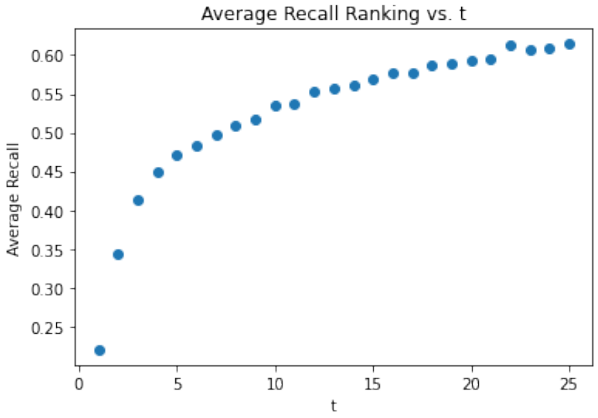


Figure 33: Recall versus t for ranking using NNMF collaborative filter.

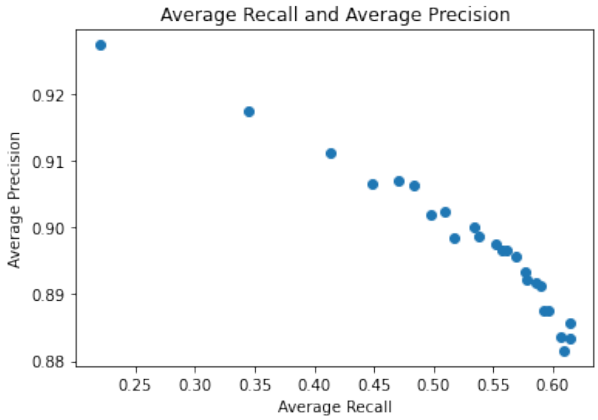


Figure 34: Precision versus Recall for ranking using NNMF collaborative filter.

Reference comments on plot shape above

7.4 Question 38

From Figure 35 we can see that Precision and t are inversely related. From Figure 36 we see that the opposite is true for Recall versus t. Figure 37 confirms this inverse relationship between Precision and Recall as t increases.

This makes sense because as we increase the set of items t recommended to the user, the denominator of the Precision function grows larger, resulting in a smaller fraction. Since Recall is independent of the size of t, it does not exhibit this behavior. Intuitively, as we increase the size of the recommended list, we are less likely for the user to like all the recommended items (Precision goes down), but more likely to recommend a movie the user likes (Recall goes up).

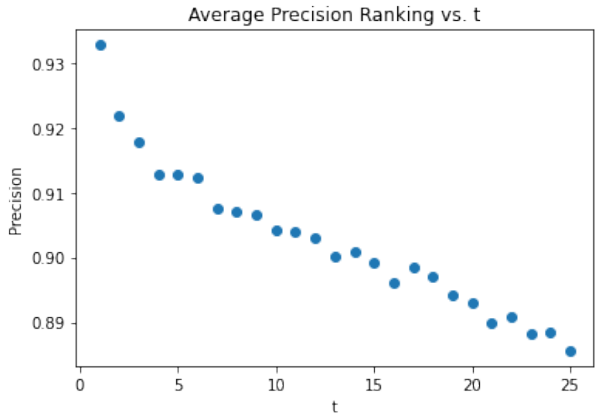


Figure 35: Precision versus t for ranking using MF with bias collaborative filter.

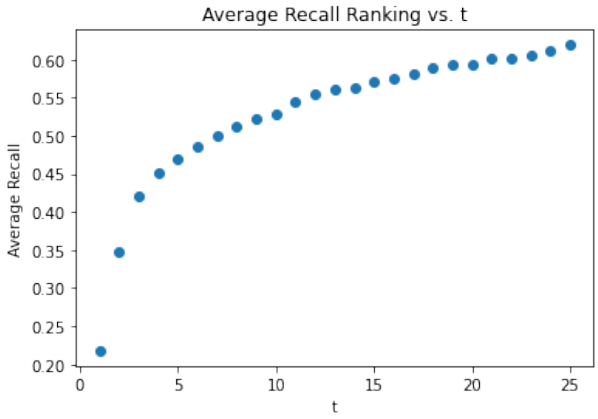


Figure 36: Recall versus t for ranking using MF with bias collaborative filter.

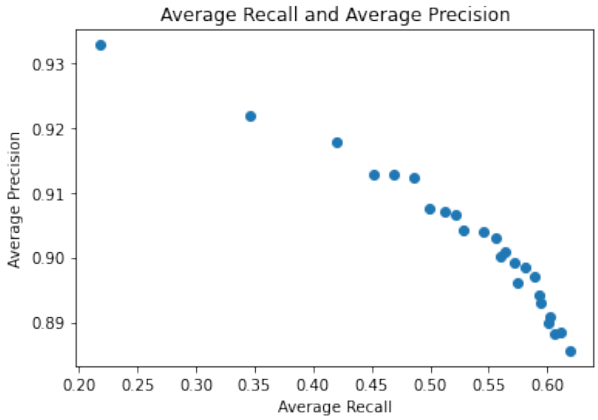


Figure 37: Precision versus Recall for ranking using MF with bias collaborative filter.

7.5 Question 39

The relevance of the recommendation list generated using k-NN, NNMF, and MF with bias predictions are pretty consistent throughout the different algorithms implemented.NNMF appears to be marginally better and converges more steadily. KNN appears to be better than MF with bias, but the overall shapes are almost indistinguishable. This may be due to each algorithm being implemented with their respective hyperparameters. This demonstrates that these algorithms can all make almost identical predictions once parameters are fine tuned.

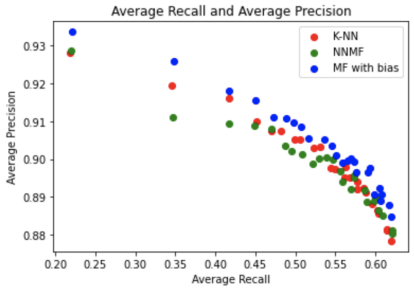


Figure 38: Precision-Recall curves for K-NN, NNMF, and MF with bias

Project 3

Import General Libraries

```
from google.colab import drive
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sys

!pip install surprise

drive.mount('/content/drive/')
sys.path.append('/content/drive/MyDrive')
```

```
Collecting surprise
  Downloading https://files.pythonhosted.org/packages/61/de/e5cba8682201fcf9c3719a6fdda
Collecting scikit-surprise
  Downloading https://files.pythonhosted.org/packages/97/37/5d334adaf5ddd65da99fc65f650
     |████████████████████████████████| 11.8MB 356kB/s
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: numpy>=1.11.2 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist-packages (f
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.1-cp37-cp37m-linux_x8
  Stored in directory: /root/.cache/pip/wheels/78/9c/3d/41b419c9d2aff5b6e2b4c0fc8d25c53
Successfully built scikit-surprise
Installing collected packages: scikit-surprise, surprise
Successfully installed scikit-surprise-1.1.1 surprise-0.1
Mounted at /content/drive/
```

MovieLens dataset

```
'''
Import dataset
'''

ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')


user_1 = ratings_df.loc[0]
print(user_1)
```

```
print(user_1)
```

```
userId               1.0
movieId              1.0
rating               4.0
timestamp    964982703.0
Name: 0, dtype: float64
```

```
'''
Create R matrix
'''

num_users = np.max(np.unique(ratings_df['userId']))
print(num_users)
num_movies = np.max(np.unique(ratings_df['movieId']))
print(num_movies)
R = np.zeros((num_users, num_movies))

# loop through the rows of ratings.df
# the 0th index of each row of ratings_df corresponds to the userId
# the 1st index corresponds to the movieId
# the 2nd index corresponds to rating
num_rows,_ = ratings_df.shape
for i in np.arange(num_rows):
    row_i = ratings_df.loc[i]
    user_id = int(row_i[0]-1)
    movie_id = int(row_i[1]-1)
    rating = row_i[2]
    R[user_id, movie_id] = rating
```

```
610
193609
```

```
print(R)
```

```
[[4.  0.  4.  ... 0.  0.  0. ]
 [0.  0.  0.  ... 0.  0.  0. ]
 [0.  0.  0.  ... 0.  0.  0. ]
 ...
 [2.5 2.  2.  ... 0.  0.  0. ]
 [3.  0.  0.  ... 0.  0.  0. ]
 [5.  0.  0.  ... 0.  0.  0. ]]
```

Question 1

```
'''
Compute sparcity of movie ratings dataset
'''

existing_ratings = len(ratings_df['rating']) #has 100836 rows / 4 columns
```

```
number_of_users = ratings_df['userId'].nunique()
num_movies = ratings_df['movieId'].nunique()
num_users = np.max(np.unique(ratings_df['userId']))

print('number of users is', num_users)
print('number of movies is',num_movies)
print('existing ratings', existing_ratings)

Sparcity = existing_ratings / (number_of_users * num_movies)


print('The sparcity level of the dataset is {}'.format((Sparcity)))
```

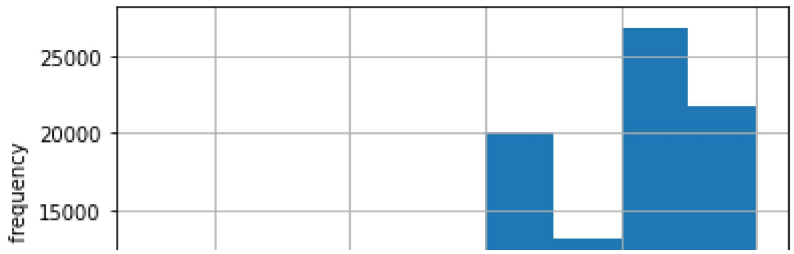
```
number of users is 610
number of movies is 9724
existing ratings 100836
The sparcity level of the dataset is 0.016999683055613623
```

```
ratings = ratings_df['rating']
movies_index= ratings_df['movieId']
print(movies_index[0])
```

```
1
```

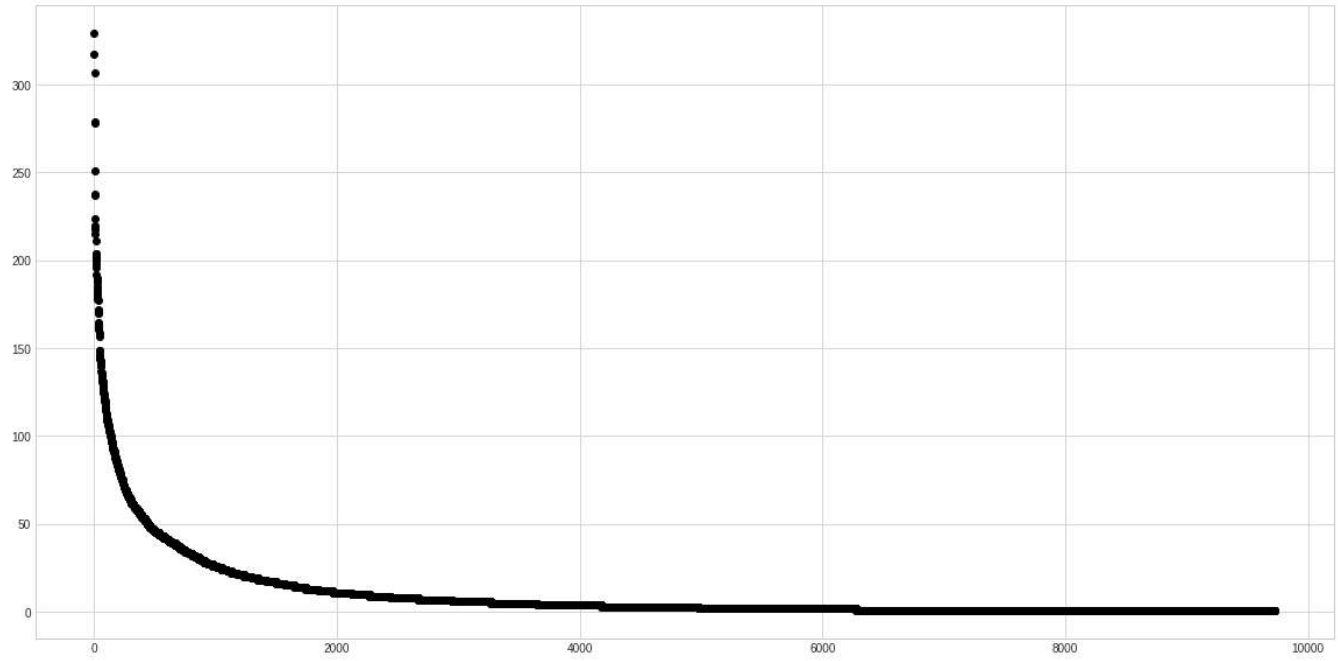
Question 2

```
'''
Plot histogram showing frequency of rating values
'''

ax = ratings_df['rating'].hist(bins=[0.5,1.0,1.5,2.0,2.5,3.0, 3.5, 4.0, 4.5, 5.0],)
ax.set_xlabel("Ratings")
ax.set_ylabel("Rating frequency")

x=ratings_df.groupby(['movieId', 'rating'])#agg(['count', 'sum'])
groups = dict(list(x))
#groups
```



Question 3

```
'''
Plot distribution of the number of rating recieved among movies
'''


number_of_ratings = np.zeros(num_movies)
mv_index = []



for i in range(0,existing_ratings):
    id  = int(movies_index[i])
    if id in mv_index:
        ind = np.where(mv_index == id)
        ind = int(ind[0])
        number_of_ratings[ind] = number_of_ratings[ind] + 1
    else:
        mv_index = np.append(mv_index,id)
        ind = np.where(mv_index == id)
        ind = int(ind[0])
        number_of_ratings[ind] = 1



#print(np.max(number_of_ratings))
number_of_ratings = number_of_ratings[0:len(mv_index)]
combined = np.column_stack((mv_index,number_of_ratings,))
a = combined
a = a[a[:,1].argsort()]
a = np.flip(a,0)



%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
plt.figure(figsize=(20,10))
plt.plot(range(len(mv_index)),a[:,1], 'o', color='black');
```

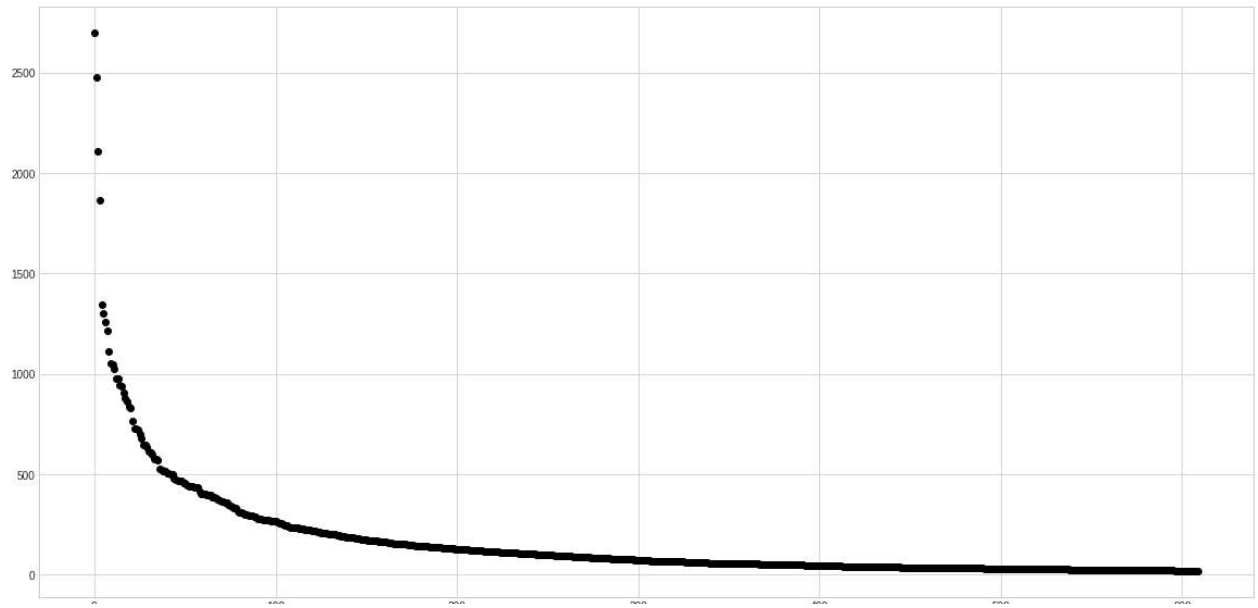



```
popular = []
unpopular = []


for i in range(0,len(a[:,0])):
    if a[i,1] > 2:
        popular.append(a[i,0])
    else:
        unpopular.append(a[i,0])


print(len(popular))
print(len(unpopular))
print(len(a[:,0]))
```

```
4980
4744
9724
```

Question 4

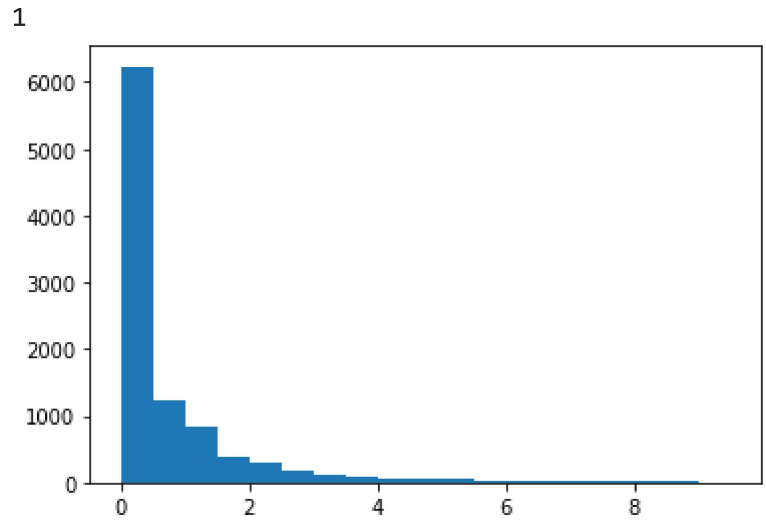
```
'''
Plot distribution of the number of rating given among users
'''



number_of_ratings = np.zeros(num_users)
us_index = []

users_index= ratings_df['userId']

for i in range(0,existing_ratings):
    id  = int(users_index[i])
    if id in us_index:
        ind = np.where(us_index == id)
        ind = int(ind[0])
        number_of_ratings[ind] = number_of_ratings[ind] + 1
    else:
        us_index = np.append(us_index,id)
        ind = np.where(us_index == id)
        ind = int(ind[0])
        number_of_ratings[ind] = 1



#print(np.max(number_of_ratings))
number_of_ratings = number_of_ratings[0:len(us_index)]
combined = np.column_stack((us_index,number_of_ratings,))
a = combined
a = a[a[:,1].argsort()]
a = np.flip(a,0)



%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
plt.figure(figsize=(20,10))
plt.plot(range(len(us_index)),a[:,1], 'o', color='black');
```



Question 6

```
'''
Plot the variance of the rating values receieved by each movie in bins
'''
from matplotlib import pyplot as plt
ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
num_movies = np.max(np.unique(ratings_df['movieId']))
#print(ratings_df)
mov_id = ratings_df['movieId']
rating = ratings_df['rating']
mean = np.zeros(num_movies+1)
num_ratings = np.zeros(num_movies+1)
temp = 0

temp = mov_id[0]
print(temp)
for i in range(0,len(mov_id)):
    temp = mov_id[i]
    mean[temp] = mean[temp] + rating[i]
    num_ratings[temp] =  num_ratings[temp] + 1


avg_mean = np.zeros(num_movies+1)
for i in range(0,len(mov_id)):
    temp = mov_id[i]
    avg_mean[temp] = mean[temp]/num_ratings[temp]
#print(len(avg_mean))
squared_diff = np.zeros(num_movies+1)
for j in range(0,len(rating)):
    temp = mov_id[j]
    temp1 = avg_mean[temp]
    temp2 = rating[j] - temp1
```

```
        temp3 = temp2*temp2
        squared_diff[j] = temp3

    sum_sq_diff = np.zeros(num_movies+1)
    for i in range(0,len(rating)):
        temp = mov_id[i]
        sum_sq_diff[temp] = sum_sq_diff[temp] + squared_diff[temp]
    #print(sum_sq_diff)
    variance = np.zeros(num_movies+1)
    only_real = []
    for j in range(0,num_movies+1):
        if num_ratings[j] != 0:
            variance[j] = sum_sq_diff[j]/num_ratings[j]
            only_real.append(variance[j])
    #print(np.max(variance))

    bins = np.arange(0, 10, 0.5)

    plt.hist(only_real,bins = bins)
    plt.show()
```



Neighborhood-based collaborative filtering

Question 10

```
'''Sweep k from 2 to 100 in steps of two, and plot rmse/mae vs k'''


from surprise.model_selection.validation import cross_validate
from surprise.prediction_algorithms.knns import KNNWithMeans
from surprise import Dataset
from surprise import Reader
```

```
from surprise import Reader
from surprise import accuracy
from surprise.model_selection import KFold
import numpy as np
import pandas as pd

from google.colab import files


import matplotlib.pyplot as plt
import time
all_results=[]
def plot_results(ks, cv_results):

    plt.xlabel('k')
    plt.ylabel('RMSE value')
    plt.title('RMSE vs K ')
    plt.plot( ks,cv_results[0,:])
   # plt.legend(['RMSE', 'MAE'])
    plt.show()

    plt.xlabel('k')
    plt.ylabel('MAE value')
    plt.title('MAE vs K ')
    plt.plot( ks, cv_results[1,:])
    plt.show()
    #'rmse is root mean square error'



def knn_cross_validation(data):

    #cv_results = np.zeros((2,50)) # need one column
    sim_options = {'name': 'pearson'}

    number_of_folds = 10
    timenow= time.time()
    idx=0
    for k in range(2,101,2):
        print('I am on ', k )
        algo = KNNWithMeans(k=k, sim_options=sim_options)

        cv = cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=number_of_folds, verbose
        cv_results[0,idx] = np.mean(cv['test_rmse'])
        cv_results[1,idx] = np.mean(cv['test_mae'])
        all_results.append(cv_results)

        a=cv_results[0, idx]
        b=cv_results[1, idx]

        print('RMSE = {}'.format((a))) #had to break it up, otherwise positional arguement er
        #print('MAE = {}'.format((b)))
```

```
            idx += 1
            # results_df =pd.DataFrame({k: [a, b]})
            # results_df.to_csv('{} results.csv'.format(k))
            # files.download('{} results.csv'.format(k))
        print('This took {} seconds'.format(time.time()-timenow))
        ks = np.arange(2,101,2)


        plot_results(ks,cv_results )

    ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
    reader = Reader(rating_scale=(0.5, 5.0))
    data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader)
    cv_results = np.zeros((2,50))
    knn_cross_validation(data)
```

```
I am on  2
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
RMSE = 1.0194218846107304
```

Question 11

```
Done computing similarity matrix.
```

```
'''Report the average RMSE and average MAE value for the minimum k found in Question 10'''

print('The RMSE value for k = 14 is {}'.format(cv_results[0,14]))
print('The MAE value for k = 14 is {}'.format(cv_results[1,14]))
```

```
The RMSE value for k = 14 is 0.8892693748711421
The MAE value for k = 14 is 0.6773293379151676
Done computing similarity matrix.
```

Question 12-14

```
Computing the pearson similarity matrix...
```

```
'''
Define functions used in Questions 24-29
'''

from surprise import Reader
from surprise import Dataset
from surprise.prediction_algorithms.matrix_factorization import SVD
from surprise.model_selection import cross_validate
from surprise.model_selection import KFold
from surprise import accuracy
import matplotlib.pyplot as plt
from surprise.model_selection.validation import cross_validate
from surprise.prediction_algorithms.knns import KNNWithMeans
from surprise import Dataset
```

```
from surprise import Dataset
from surprise import Reader
from surprise import accuracy
from surprise.model_selection import KFold
import numpy as np
import pandas as pd

class MF_Bias:
    '''
    Class to design an 10-fold cross validation using MF w/ bias collaborative
    filtering over different values of k
    '''

    def __init__(self, ratings_df, k, trim=False):
        '''
        data: data this is performed on; a DataFrame
        k: a list storing the values of k
        '''
        if trim==True:
            self.ratings_df = ratings_df

            # count number of ratings
            self.num_ratings = count_movie_ratings(self.ratings_df)

            # define popular movies: movies that have recieved more than 2 ratings
            self.popular = np.where(self.num_ratings > 2)[0]

            # define high-variance movies: movies that have recieved at least 5
            # ratings and have variance of rating values recieved of at least 2
            var_movies = count_movie_variability(self.ratings_df)
            self.high_var = []
            for i in np.where(self.num_ratings >= 5)[0]:
                if len(np.unique(var_movies[i:])) >= 2:
                    self.high_var.append(i)

        # load data in format that SVD can use
        reader = Reader(rating_scale=(1.0,5.0))
        self.data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']],
                                         reader=reader)
        self.k = k


    def trim_popular(self, testset):
        '''
        Trims testset so that only movies in popular and unpopular trimmed
        subsets movies remain

        Returns trimmed testset lists
        '''
        pop_test = []
        unpop_test = []
        for i in np.arange(len(testset)):
```

```
            for j in self.popular:
                # if movie is in the popular subset
                if j == testset[i][1]:
                    pop_test.append(testset[i])
                    break
                # since subset_movies is ordered from least to greatest,
                # if j > testset[i][1], then we know testset[i][1] is not in
                # popular set, so therefore it is in the unpopluar set
                elif j > testset[i][1]:
                    unpop_test.append(testset[i])
                    break
        return pop_test, unpop_test


    def trim_var(self, testset):
        '''
        Trims testset so that only movies in high variance trimmed movies subsets
        movies remain

        Returns trimmed testset lists
        '''
        var_test = []
        for i in np.arange(len(testset)):
            for j in np.arange(len(self.high_var)):
                # if movie is in the popular subset
                if j == testset[i][1]:
                    var_test.append(testset[i])
                    break
                # since subset_movies is ordered from least to greatest,
                # if j > testset[i][1], then we know testset[i][1] is not in
                # high variance data set
                elif j > testset[i][1]:
                    break
        return var_test


    def predict(self):
        '''
        Performs 10-fold cv over different values of k for full data set

        Returns average RMSE and MAE for each value of k
        '''
        rmse = np.zeros(len(self.k))
        mae = np.zeros(len(self.k))
        for i in np.arange(len(k)):
            print(k[i])
            sim_options = {'name': 'pearson'}
            algo = KNNWithMeans(k=k[i], sim_options=sim_options)
            mf_bias = cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=10, verbose=Fal
            #mf_bias = cross_validate(algo=SVD(n_factors=k[i], biased=True),
                           # data=self.data, cv=10, n_jobs=-1)
            rmse[i] = mf_bias['test_rmse'].mean()
```

```
            mae[i] = mf_bias['test_mae'].mean()

        return rmse, mae


    def predict_trim(self):
        '''
        Trains the MF w/ bias collaborative filter across 10-folds for different
        values of k and evaluates performance on popular, unpopular, and high
        variance validation sets

        Returns average RMSE across folds for all trimmed test sets
        '''

        rmse_pop = np.zeros(len(self.k))
        rmse_unpop = np.zeros(len(self.k))
        rmse_var = np.zeros(len(self.k))
        sim_options = {'name': 'pearson'}
        for i in np.arange(len(self.k)):
            print(k[i])

            # split data in 10 folds
            kf = KFold(n_splits=10)
            # define MF w/ bias collaborative filter

            mf_bias = KNNWithMeans(k=k[i], sim_options=sim_options)
            #mf_bias = SVD(n_factors=k[i], biased=True)



            total_RMSE_pop = 0
            total_RMSE_unpop = 0
            total_RMSE_var = 0
            for trainset, testset in kf.split(self.data):

                # train and test algorithm
                mf_bias.fit(trainset)
                pop_test, unpop_test = self.trim_popular(testset)
                var_test = self.trim_var(testset)
                pred_pop = mf_bias.test(pop_test)
                pred_unpop = mf_bias.test(unpop_test)
                pred_var = mf_bias.test(var_test)

                # compute RMSEs for fold
                total_RMSE_pop += accuracy.rmse(pred_pop, verbose=False)
                total_RMSE_unpop += accuracy.rmse(pred_unpop, verbose=False)
                total_RMSE_var += accuracy.rmse(pred_var, verbose=False)

            rmse_pop[i] = total_RMSE_pop / 10
            rmse_unpop[i] = total_RMSE_unpop / 10
            rmse_var[i] = total_RMSE_var / 10
```

```
            print("Popular RMSE: ", rmse_pop[i])
            print("Unpopular RMSE: ", rmse_unpop[i])
            print("High variation RMSE: ", rmse_var[i])
            print()

        return rmse_pop, rmse_unpop, rmse_var


def plot_rmse_mae(rmse, mae=None):
    '''
    Plots RMSE and MAE as a function of k
    '''

    if mae is not None:
        # Plot MAE as a function of k
        plt.plot(k, mae)
        plt.title("k versus MAE")
        plt.xlabel("k")
        plt.xticks(k)
        plt.ylabel("MAE")
        plt.show()

    # Plot RMSE as a function of k
    plt.plot(k, rmse)
    plt.title("k versus RMSE")
    plt.xlabel("k")
    plt.xticks(k)
    plt.ylabel("RSME")
    plt.show()


def report_lowest_RMSE(rmse, k):
    '''
    Returns minimum average RMSE value
    '''
    print("Minimum RMSE value: ", np.min(rmse))
    print("at k: ", k[np.argmin(rmse)])
    return np.min(rmse)

def report_lowest_MAE(mae, k):
    '''
     Returns minimum average MAE value
    '''
    print("Minimum MAE value: ", np.min(mae))
    print("at k: ", k[np.argmin(mae)])
    return np.min(mae)



def count_movie_ratings(ratings_df):
    '''
```

```
    Counts the number of times each movie was rated

    Returns an array where the index corresonds to the movieId and the value of
    the index corresponds to the number of times the movie was rated
    '''
    num_movies = np.max(np.unique(ratings_df['movieId']))
    num_ratings = np.zeros(num_movies)
    # loop through the rows of ratings.df and count the number of times each
    # movie was rated
    for i in np.arange(ratings_df.shape[0]):
        row_i = ratings_df.loc[i]
        movie_id = int(row_i[1]-1)
        num_ratings[movie_id] += 1
    return num_ratings


def count_movie_variability(ratings_df):
    '''
    Counts the variability in movie ratings

    Returns an array where the index corresponds to the movieId and the value of
    each column corresponds to the number of times the movie was given
    rating = (column # +1)
    '''
    num_movies = np.max(np.unique(ratings_df['movieId']))
    var_ratings = np.zeros((num_movies, 5))
    # loop through the rows of ratings.df and count the number of times each
    # movie was given a certain ratings
    for i in np.arange(ratings_df.shape[0]):
        row_i = ratings_df.loc[i]
        movie_id = int(row_i[1]-1)
        rating = int(row_i[2])
        var_ratings[movie_id, rating-1] += 1
    return var_ratings
     Done computing similarity matrix.
'''
Perform 10-fold cross-validation using a knn collaborative filer on the
populur, unpopular, and high variance movie subsets
'''


# load data
ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')


# compute RMSE as a function of k for each trimmed data set
k = np.arange(2, 51, 2)
mfb = MF_Bias(ratings_df, k, trim=True)
rmse_pop, rmse_unpop, rmse_var = mfb.predict_trim()

     2
     Computing the pearson similarity matrix...
```

```
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Popular RMSE:  1.0244899559981
Unpopular RMSE:  1.0121621678439705
High variation RMSE:  1.039135146726515

4
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Popular RMSE:  0.9424043938139146
Unpopular RMSE:  0.9449173944922897
High variation RMSE:  0.9546605653614705

6
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
```

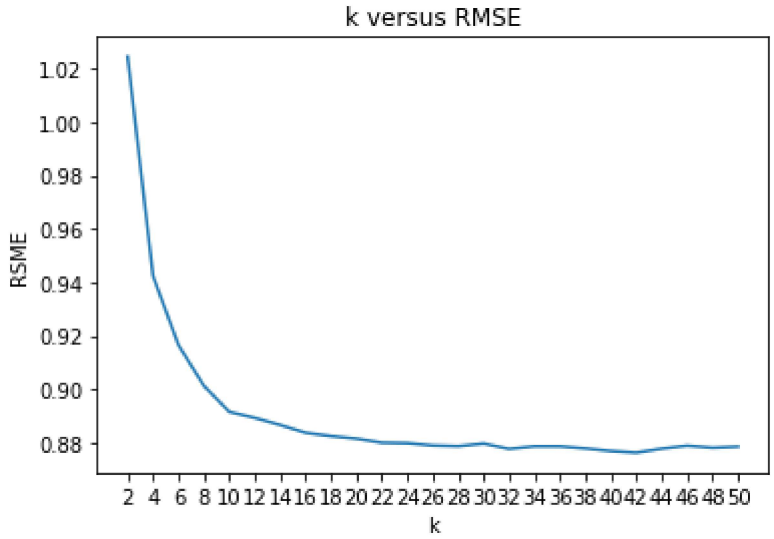
```
Computing the pearson similarity matrix...
Done computing similarity matrix.
```

```
print("RMSE for Popular Movies")
plot_rmse_mae(rmse_pop)
print()
print()
print("Minimum avg RMSE: ", report_lowest_RMSE(rmse_pop, k))
```

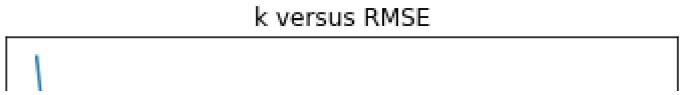
```
print("RMSE for Unpopular Movies")
plot_rmse_mae(rmse_unpop)
print()
print()
print("Minimum avg RMSE: ", report_lowest_RMSE(rmse_unpop, k))
```

```
print("RMSE for High Variance Movies")
plot_rmse_mae(rmse_var)
print()
print()
print("Minimum avg RMSE: ", report_lowest_RMSE(rmse_var, k))
```

```
RMSE for Popular Movies
```




```
Minimum RMSE value:  0.8763318701621925
at k:  42
Minimum avg RMSE:  0.8763318701621925
RMSE for Unpopular Movies
```

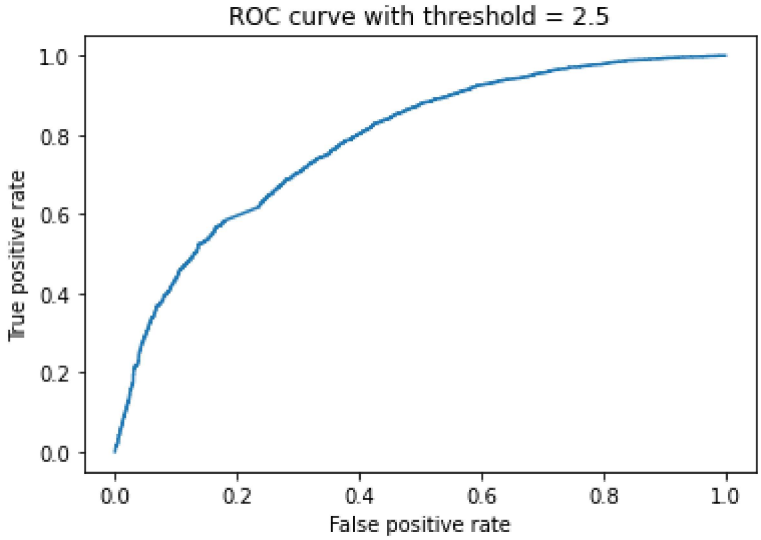


Question 15

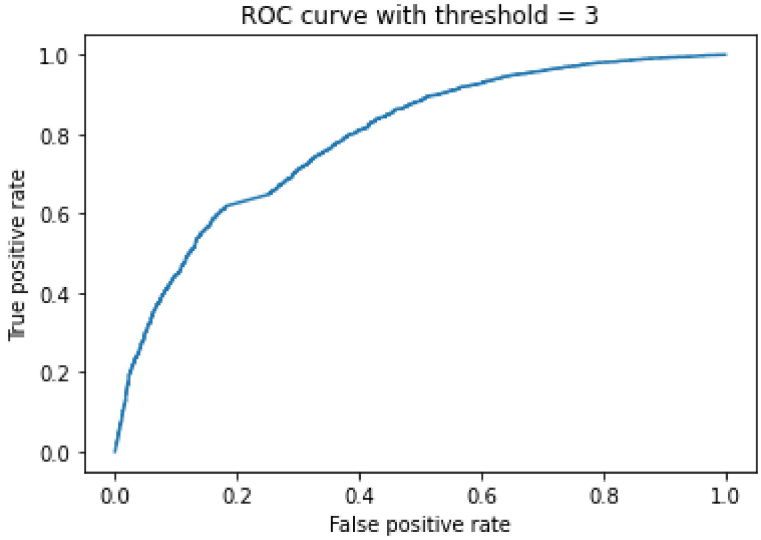
```
'''
Plot the ROC for the K-NN collaborative filter for the threshold values [ 2.5, 3, 3.5, 4]

For each of the plots, also report the area under the curve
''''



from surprise.model_selection.validation import cross_validate
from surprise.prediction_algorithms.knns import KNNWithMeans
from surprise import Dataset
from surprise import Reader
from surprise import accuracy
from surprise.model_selection import KFold
import numpy as np
import pandas as pd
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from surprise.model_selection.split import train_test_split


threshold_vals = [ 2.5, 3, 3.5, 4]

ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
```

```
    reader = Reader(rating_scale=(0.5, 5.0))
    data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader)

    sim_options={'name': 'pearson'}
    for threshold in threshold_vals:
        algo = KNNWithMeans(k=14, sim_options={'name': 'pearson'})
        trainset, testset = train_test_split(data, test_size=0.1)
        algo.fit(trainset)
        pred = algo.test(testset)
        print(testset[0])
        print(pred)
        y_true = []
        y_score = []
        #each is a column for the testset
        ###checking the  rating against the threshold.
        for i in range(len(pred)):
            y_score.append(pred[i].est)

            if testset[i][2] >= threshold: #ratings threshold
                y_true.append(1)
            else:
                y_true.append(0)


        #print('this', y_true)
        fpr, tpr, thresholds = roc_curve(y_true, y_score)
        auc_ = roc_auc_score(y_true,  y_score)

        plt.plot(fpr,tpr)
        plt.xlabel('False positive rate')
        plt.ylabel('True positive rate')
        plt.title('ROC curve with threshold = {}'.format(threshold))
        plt.show()
        print('The area under the curve (AUC) value is {}'.format(auc_))
```

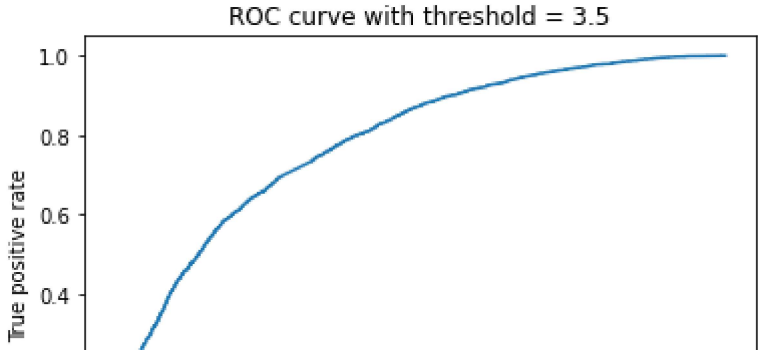
```
Computing the pearson similarity matrix...
Done computing similarity matrix.
(607, 3016, 4.0)
[Prediction(uid=607, iid=3016, r_ui=4.0, est=3.4115294518966297, details={'actual_k': 8
```

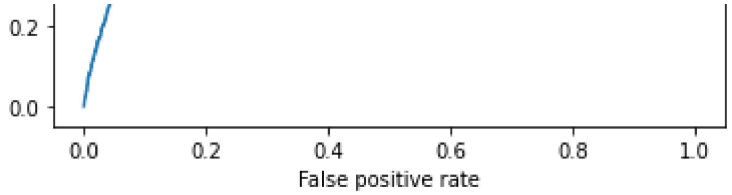



```
The area under the curve (AUC) value is 0.783143272653317
Computing the pearson similarity matrix...
Done computing similarity matrix.
(274, 1370, 3.5)
[Prediction(uid=274, iid=1370, r_ui=3.5, est=3.17455592896545, details={'actual_k': 14,
```




```
The area under the curve (AUC) value is 0.7908626570017433
Computing the pearson similarity matrix...
Done computing similarity matrix.
(47, 2028, 4.0)
[Prediction(uid=47, iid=2028, r_ui=4.0, est=3.6540885887938495, details={'actual_k': 14
```






```
The area under the curve (AUC) value is 0.7838432232467198
Computing the pearson similarity matrix...
Done computing similarity matrix.
(608, 2571, 5.0)
[Prediction(uid=608, iid=2571, r_ui=5.0, est=3.61696378270811, details={'actual_k': 14,
```

Model-based collaborative filtering

Non-negative matrix factorization (NNMF)

Question 17

```
from surprise.prediction_algorithms.matrix_factorization import NMF
from surprise.model_selection import cross_validate
import matplotlib.pyplot as plt
from surprise.prediction_algorithms.matrix_factorization import NMF
from surprise.model_selection import cross_validate
import matplotlib.pyplot as plt
from surprise import Reader
from surprise import Dataset

class NNMF:
    '''
    Class to design a 10-fold cross validation using NNMF collaborative
    filtering over different values of k
    '''
    def __init__(self, data, k):

        '''
        data: data this is performed on; a DataFrame
        k: a list storing the values of k
        '''

        self.data = data
        self.k = k
        self.poprmse = np.zeros(len(k))
        self.popmae = np.zeros(len(k))
        self.unpoprmse = np.zeros(len(k))
        self.unpopmae = np.zeros(len(k))
        self.highvarrmse = np.zeros(len(k))
        self.highvarmae = np.zeros(len(k))
```

```
    def cv_over_k_pop(self):
        '''
        Performs 10-fold cv over different values of k for popular trimmed dataset
        '''

        from surprise.model_selection.split import KFold
        from surprise import accuracy
        ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
        reader = Reader(rating_scale=(0.5,5.0))
        train_data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader
        kf = KFold(n_splits=10)
        k = self.k

        for i in range(0,len(k)):
            print(i)
            popmae = 0
            poprmse = 0

            print(k[i])
            n_factors = k[i]
            inc = 1
            for trainset, testset in kf.split(train_data):
                print(inc)
                inc = inc+1
                nnmf = NMF(n_factors=n_factors, biased=False)
                nnmf.fit(trainset)
                pop_test = []
                unpop_test = []
                ## these have weird variables because of contradictions
                for j in range(0,len(testset)):
                    for o in popular:
                        if o == testset[j][1]:
                            #print(testset[j][1])
                            #print(i)
                            pop_test.append(testset[j][:])
                #print(len(testset))
                #print(len(pop_test))
                #print(len(unpop_test))
                poppredictions = nnmf.test(pop_test)
                poprmse = poprmse + accuracy.rmse(poppredictions,verbose = False)
                popmae = popmae + accuracy.mae(poppredictions,verbose = False)
            self.poprmse[i] = poprmse/10
            self.popmae[i] = popmae/10


    def cv_over_k_unpop(self):

        '''
        Performs 10-fold cv over different values of k for unpopular trimmed dataset
        '''
```

```
        from surprise.model_selection.split import KFold
        from surprise import accuracy
        ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
        reader = Reader(rating_scale=(0.5,5.0))
        train_data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader
        kf = KFold(n_splits=10)
        k = self.k

        for i in range(0,len(k)):
            print(i)

            unpopmae = 0
            unpoprmse = 0
            print(k[i])
            n_factors = k[i]
            inc = 1
            for trainset, testset in kf.split(train_data):
                print(inc)
                inc = inc+1
                nnmf = NMF(n_factors=n_factors, biased=False)
                nnmf.fit(trainset)

                unpop_test = []
                ## these have weird variables because of contradictions
                for j in range(0,len(testset)):
                    for l in unpopular:
                        if l == testset[j][1]:
                            #print(testset[j][1])
                            #print(i)
                            unpop_test.append(testset[j][:])
                #print(len(testset))
                #print(len(pop_test))
                #print(len(unpop_test))
                unpoppredictions = nnmf.test(unpop_test)
                unpoprmse = unpoprmse + accuracy.rmse(unpoppredictions,verbose = False)
                unpopmae = unpopmae + accuracy.mae(unpoppredictions,verbose = False)
            self.unpoprmse[i] = unpoprmse/10
            self.unpopmae[i] = unpopmae/10

    def cv_over_k_highvar(self):

        '''
        Performs 10-fold cv over different values of k for high variance trimmed dataset
        '''

        from surprise.model_selection.split import KFold
        from surprise import accuracy
        ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
        reader = Reader(rating_scale=(0.5,5.0))
        train_data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader
        kf = KFold(n_splits=10)
```

```
        k = self.k

        for i in range(0,len(k)):
            print(i)

            highvarmae = 0
            highvarrmse = 0
            print(k[i])
            n_factors = k[i]
            inc = 1
            for trainset, testset in kf.split(train_data):
                print(inc)
                inc = inc+1
                nnmf = NMF(n_factors=n_factors, biased=False)
                nnmf.fit(trainset)

                highvar_test = []
                ## these have weird variables because of contradictions
                for j in range(0,len(testset)):
                    for l in high_var:
                        if l == testset[j][1]:
                            #print(testset[j][1])
                            #print(i)
                            highvar_test.append(testset[j][:])
                #print(len(testset))
                #print(len(pop_test))
                #print(len(unpop_test))
                highvarpredictions = nnmf.test(highvar_test)
                highvarrmse = highvarrmse + accuracy.rmse(highvarpredictions,verbose = False)
                highvarmae = highvarmae + accuracy.mae(highvarpredictions,verbose = False)
            self.highvarrmse[i] = highvarrmse/10
            self.highvarmae[i] = highvarmae/10


    def plot_rmse_mae_pop(self):

        '''
        Plots RMSE and MAE as a function of k for popular trimmed dataset
        '''

        # Plot RMSE as a function of k
        plt.plot(self.k, self.poprmse)
        plt.title("k versus RMSE for Popular Dataset for NNMF")
        plt.xlabel("k")
        plt.xticks(self.k)
        plt.ylabel("RSME")
        plt.show()

        # Plot MAE as a function of k
        plt.plot(self.k, self.popmae)
        plt.title("k versus MAE for Popular Dataset for NNMF")
        plt.xlabel("k")
```

```
        plt.xticks(self.k)
        plt.ylabel("MAE")
        plt.show()


    def plot_rmse_mae_unpop(self):

        '''
        Plots RMSE and MAE as a function of k for unpopular trimmed dataset
        '''



        # Plot RMSE as a function of k
        plt.plot(self.k, self.unpoprmse)
        plt.title("k versus RMSE for Unpopular Datasetfor NNMF")
        plt.xlabel("k")
        plt.xticks(self.k)
        plt.ylabel("RSME")
        plt.show()

        # Plot MAE as a function of k
        plt.plot(self.k, self.unpopmae)
        plt.title("k versus MAE for Unpopular Dataset for NNMF")
        plt.xlabel("k")
        plt.xticks(self.k)
        plt.ylabel("MAE")
        plt.show()


    def plot_rmse_mae_highvar(self):

        '''
        Plots RMSE and MAE as a function of k for high variance trimmed dataset
        '''

        # Plot RMSE as a function of k
        plt.plot(self.k, self.highvarrmse)
        plt.title("k versus RMSE for High Variance Dataset for NNMF")
        plt.xlabel("k")
        plt.xticks(self.k)
        plt.ylabel("RSME")
        plt.show()

        # Plot MAE as a function of k
        plt.plot(self.k, self.highvarmae)
        plt.title("k versus MAE for High Variance Dataset for NNMF")
        plt.xlabel("k")
        plt.xticks(self.k)
        plt.ylabel("MAE")
        plt.show()
```

```
    def report_lowest_RMSE_pop(self):
        '''
        Returns minimum average RMSE value for popular trimmed dataset
        '''
        return np.amin(self.poprmse)


    def report_lowest_MAE_pop(self):
        '''
        Returns minimum average MAE value for popular trimmed dataset
        '''
        return np.amin(self.popmae)

    def report_lowest_RMSE_unpop(self):
        '''
        Returns minimum average RMSE value for unpopular trimmed dataset
        '''
        return np.amin(self.unpoprmse)


    def report_lowest_MAE_unpop(self):
        '''
        Returns minimum average MAE value for unpopular trimmed dataset
        '''
        return np.amin(self.unpopmae)

    def report_lowest_RMSE_highvar(self):
        '''
        Returns minimum average RMSE value for high variance trimmed dataset
        '''
        return np.amin(self.highvarrmse)


    def report_lowest_MAE_highvar(self):
        '''
        Returns minimum average MAE value for high variance trimmed dataset
        '''


        return np.amin(self.highvarmae)
```

```
'''
Perform 10-fold cross-validation using an NNMF collaborative filter
and print RMSE and MAE as a function of k
'''

from surprise import Reader
from surprise import Dataset
from surprise.model_selection.split import KFold
from surprise import accuracy
ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
reader = Reader(rating_scale=(0.5,5.0))
```

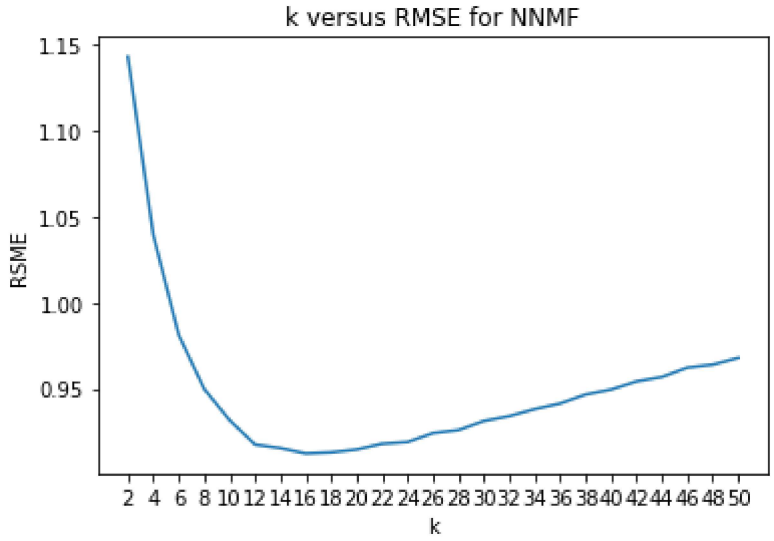
```
train_data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader=reader)
kf = KFold(n_splits=10)
k = np.arange(2, 51, 2)
rmse = np.zeros(len(k))
mae = np.zeros(len(k))
for i in range(0,len(k)):


    mae_temp = 0
    rmse_temp = 0
    print(k[i])
    n_factors = k[i]

    for trainset, testset in kf.split(train_data):

        inc = inc+1
        nnmf = NMF(n_factors=n_factors, biased=False)
        nnmf.fit(trainset)

        predictions = nnmf.test(testset)
        rmse_temp = rmse_temp + accuracy.rmse(predictions,verbose = False)
        mae_temp = mae_temp + accuracy.mae(predictions,verbose = False)
    rmse[i] = rmse_temp/10
    mae[i] = mae_temp/10
```

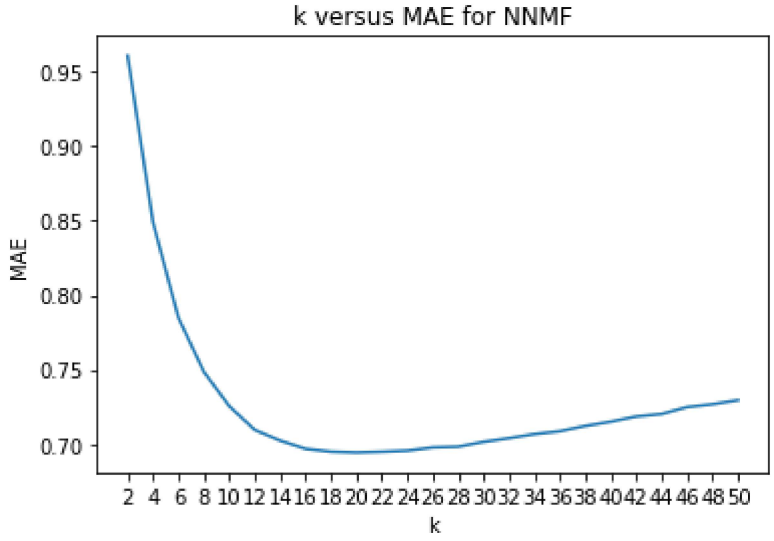
```
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
```

```
RMSE = 0.8906005570084975
```

```
# Plot RMSE as a function of k
plt.plot(k, rmse)
plt.title("k versus RMSE for NNMF")
```

```
plt.xlabel("k")
plt.xticks(k)
plt.ylabel("RSME")
plt.show()

# Plot MAE as a function of k
plt.plot(k, mae)
plt.title("k versus MAE for NNMF")
plt.xlabel("k")
plt.xticks(k)
plt.ylabel("MAE")
plt.show()
```






```
Computing the pearson similarity matrix...
```

Question 18

```
Computing the pearson similarity matrix...

'''

Minimum RMSE and MAE
'''
```

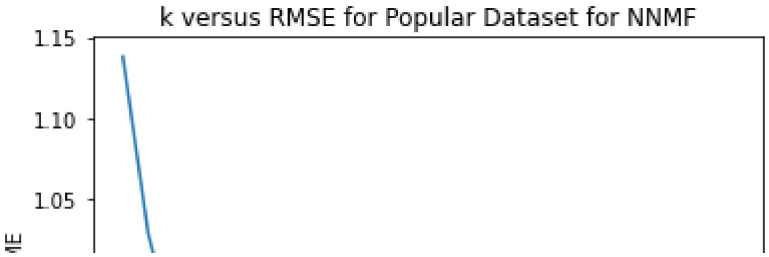
```
print("Minimum avg RMSE: ", np.amin(rmse))
print("Minimum avg MAE: ", np.amin(mae))
print("Optimal k: ", k[np.where(mae == np.amin(mae))[0]])
```

```
Minimum avg RMSE:  0.9123938286631338
Minimum avg MAE:  0.694289614514209
Optimal k:  [20]
```

Question 19

```
'''
Perform 10-fold cross-validation using a NNMF collaborative filter
and print RMSE and MAE as a function of k on popular dataset
'''

k = np.arange(2, 51, 2)
ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
reader = Reader(rating_scale=(0.5,5.0))
data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader=reader)
nn = NNMF(data, k)
nn.cv_over_k_pop()

nn.plot_rmse_mae_pop()

print("Minimum avg RMSE: ", nn.report_lowest_RMSE_pop())
print("Minimum avg MAE: ", nn.report_lowest_MAE_pop())
```

```
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
Minimum avg RMSE:  0.8913882930927883
Minimum avg MAE:  0.6772573220180013
```

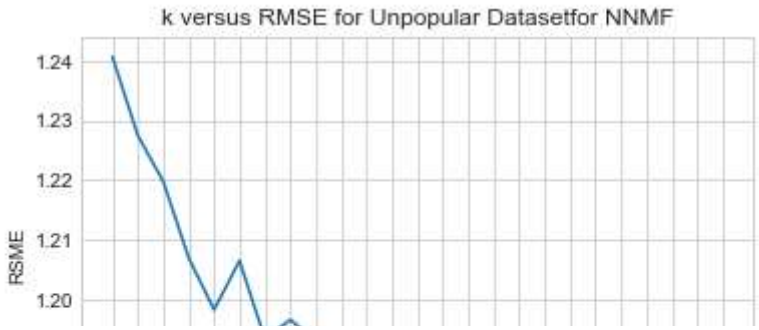


Question 20

```
'''
Perform 10-fold cross-validation using a NNMF collaborative filter
and print RMSE and MAE as a function of k on popular dataset
'''

k = np.arange(2, 51, 2)
ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
reader = Reader(rating_scale=(0.5,5.0))
data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader=reader)
nn = NNMF(data, k)
nn.cv_over_k_unpop()
print("Minimum avg RMSE: ", nn.report_lowest_RMSE_unpop())
print("Minimum avg MAE: ", nn.report_lowest_MAE_unpop())
nn.plot_rmse_mae_unpop()

print("Minimum avg RMSE: ", nn.report_lowest_RMSE_unpop())
```

```
print("Minimum avg MAE: ", nn.report_lowest_MAE_unpop())
```

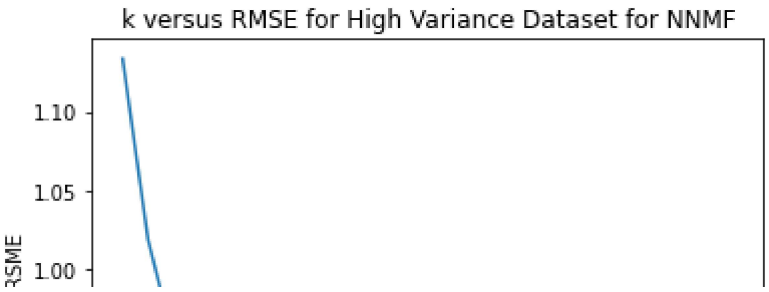
```
0
2
1
2
3
4
5
6
7
8
9
10
1
4
1
2
3
4
5
6
```



Question 21

```
'''
Perform 10-fold cross-validation using a NNMF collaborative filter
and print RMSE and MAE as a function of k on popular dataset
'''

k = np.arange(2, 51, 2)
ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
reader = Reader(rating_scale=(0.5,5.0))
data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader=reader)
nn = NNMF(data, k)
nn.cv_over_k_highvar()
print("Minimum avg RMSE: ", nn.report_lowest_RMSE_highvar())
print("Minimum avg MAE: ", nn.report_lowest_MAE_highvar())
nn.plot_rmse_mae_highvar()

print("Minimum avg RMSE: ", nn.report_lowest_RMSE_highvar())
print("Minimum avg MAE: ", nn.report_lowest_MAE_highvar())
```

```
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40
42
44
46
48
50
Minimum avg RMSE:  0.8827169869620324
Minimum avg MAE:  0.6726525333052182
```

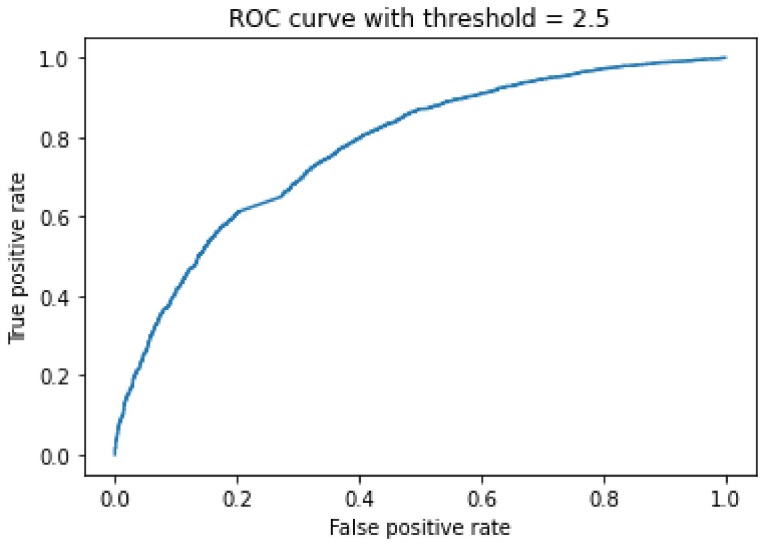


Question 22

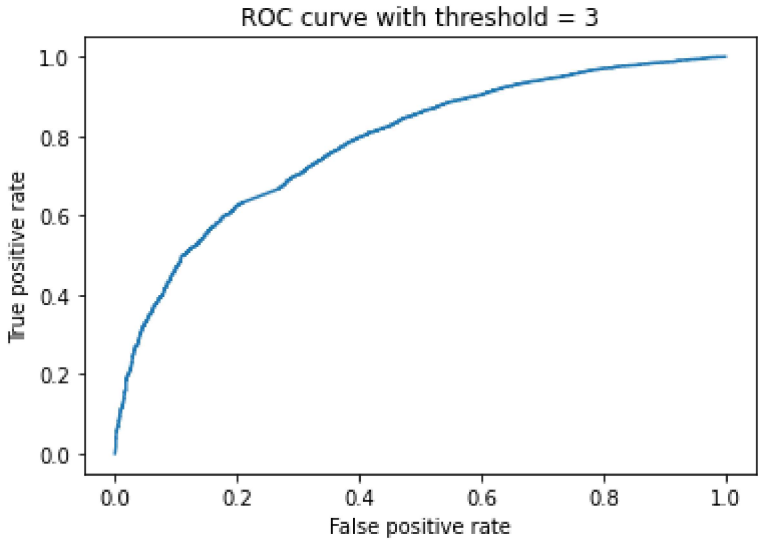
```
'''
Plot the ROC for the NNMF collaborative filter for the threshold values [ 2.5, 3, 3.5, 4]

For each of the plots, also report the area under the curve
'''



from surprise.model_selection.validation import cross_validate
from surprise.prediction_algorithms.matrix_factorization import NMF
from surprise import Dataset
from surprise import Reader
from surprise import accuracy
from surprise.model_selection import KFold
import numpy as np
import pandas as pd
```

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from surprise.model_selection.split import train_test_split


threshold_vals = [ 2.5, 3, 3.5, 4]

ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
reader = Reader(rating_scale=(0.5, 5.0))
data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader)


for threshold in threshold_vals:
    algo = NMF(n_factors = 20, biased = False)
    trainset, testset = train_test_split(data, test_size=0.1)
    algo.fit(trainset)
    pred = algo.test(testset)
    print(testset[0])
    print(pred)
    y_true = []
    y_score = []
    #each is a column for the testset
    ###checking the  rating against the threshold.
    for i in range(len(pred)):
        y_score.append(pred[i].est)

        if testset[i][2] >= threshold: #ratings threshold
            y_true.append(1)
        else:
            y_true.append(0)



    fpr, tpr, thresholds = roc_curve(y_true, y_score)
    auc_ = roc_auc_score(y_true,  y_score)

    plt.plot(fpr,tpr)
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.title('ROC curve with threshold = {}'.format(threshold))
    plt.show()
    print('The area under the curve (AUC) value is {}'.format(auc_))
```

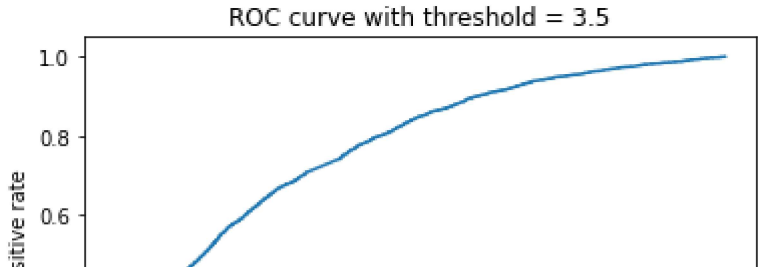
```
(274, 70286, 3.5)
[Prediction(uid=274, iid=70286, r_ui=3.5, est=3.824643405085559, details={'was_impossib
```




```
The area under the curve (AUC) value is 0.7745466968431842
(187, 1241, 4.5)
[Prediction(uid=187, iid=1241, r_ui=4.5, est=3.3695797577820272, details={'was_impossib
```




```
The area under the curve (AUC) value is 0.7829281000033087
(387, 1783, 3.0)
[Prediction(uid=387, iid=1783, r_ui=3.0, est=3.015524794837857, details={'was_impossibl
```



Question 23

```
'''
Find genre of top movies in the first 3 columns of the movie latent factor matrix
'''
```

```
from surprise.prediction_algorithms.matrix_factorization import NMF
from surprise.model_selection import cross_validate
import matplotlib.pyplot as plt
from surprise.prediction_algorithms.matrix_factorization import NMF
from surprise.model_selection import cross_validate
import matplotlib.pyplot as plt
from surprise import Reader
from surprise import Dataset

ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
reader = Reader(rating_scale=(0.5,5.0))
train_data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader=reader)
nnmf = NMF(n_factors=20, biased=False)
trainset = train_data.build_full_trainset()
nnmf.fit(trainset)
user = nnmf.pu
item = nnmf.qi

total_ids = ratings_df['movieId']
ordered_ids = []
for i in range(0,len(total_ids)):
    if total_ids[i] not in ordered_ids:
        ordered_ids.append(total_ids[i])
#print(len(ordered_ids))
ids_item_col_1 = np.zeros((len(ordered_ids),2))
ids_item_col_1[:,0] = ordered_ids[:]
ids_item_col_1[:,1] = item[:,0]
sorted_col1 = np.flip(ids_item_col_1[np.argsort(ids_item_col_1[:, 1])],0)

ids_item_col_2 = np.zeros((len(ordered_ids),2))
ids_item_col_2[:,0] = ordered_ids[:]
ids_item_col_2[:,1] = item[:,1]
sorted_col2 = np.flip(ids_item_col_2[np.argsort(ids_item_col_2[:, 1])],0)

ids_item_col_3 = np.zeros((len(ordered_ids),2))
ids_item_col_3[:,0] = ordered_ids[:]
ids_item_col_3[:,1] = item[:,2]
sorted_col3 = np.flip(ids_item_col_3[np.argsort(ids_item_col_3[:, 1])],0)
movies_df = pd.read_csv('/content/drive/MyDrive/movies.csv')
movieId = movies_df['movieId']
genre = movies_df['genres']
print('Genres for top 10 movies in column 1 \n')
for i in range(0,10):
    print(genre[np.where(movieId == sorted_col1[i,0])[0]])

print('\n \n \n Genres for top 10 movies in column 2 \n')
for i in range(0,10):
    print(genre[np.where(movieId == sorted_col2[i,0])[0]])

print('\n \n \n Genres for top 10 movies in column 3 \n')
for i in range(0,10):
```

```
print(genre[np.where(movieId == sorted_col3[i,0])[0]])

 Genres for top 10 movies in column 1

 1533    Drama|Fantasy|Mystery
 Name: genres, dtype: object
 2423    Drama
 Name: genres, dtype: object
 683    Comedy|Musical
 Name: genres, dtype: object
 6539    Action|Comedy|Crime|Thriller
 Name: genres, dtype: object
 1937    Action|Crime
 Name: genres, dtype: object
 2208    Comedy|Romance
 Name: genres, dtype: object
 1482    Comedy
 Name: genres, dtype: object
 726    Comedy|Drama|War
 Name: genres, dtype: object
 7857    Comedy
 Name: genres, dtype: object
 1898    Drama
 Name: genres, dtype: object



  Genres for top 10 movies in column 2

 7114    Fantasy|Horror
 Name: genres, dtype: object
 2829    Comedy
 Name: genres, dtype: object
 8661    Adventure|Children|Comedy
 Name: genres, dtype: object
 3688    Comedy
 Name: genres, dtype: object
 7584    Comedy|Documentary
 Name: genres, dtype: object
 7678    Drama
 Name: genres, dtype: object
 3666    Adventure|Children|Comedy
 Name: genres, dtype: object
 5867    Drama|War
 Name: genres, dtype: object
 8227    Comedy|Drama
 Name: genres, dtype: object
 3862    Comedy
 Name: genres, dtype: object



  Genres for top 10 movies in column 3

 2768    Film-Noir|Horror|Mystery|Thriller
 Name: genres, dtype: object
 1125    Comedy|Romance
```

```
Name: genres, dtype: object
4491    Drama
Name: genres, dtype: object
4009    Comedy|Drama|Romance|Thriller
```

Matrix factorization with bias

```
'''
Define functions used in Questions 24-29
'''

from surprise import Reader
from surprise import Dataset
from surprise.prediction_algorithms.matrix_factorization import SVD
from surprise.model_selection import cross_validate
from surprise.model_selection import KFold
from surprise import accuracy
import matplotlib.pyplot as plt


class MF_Bias:
    '''
    Class to design an 10-fold cross validation using MF w/ bias collaborative
    filtering over different values of k
    '''

    def __init__(self, ratings_df, k, trim=False):
        '''
        data: data this is performed on; a DataFrame
        k: a list storing the values of k
        '''
        if trim=True:
            self.ratings_df = ratings_df

            # count number of ratings
            self.num_ratings = count_movie_ratings(self.ratings_df)

            # define popular movies: movies that have recieved more than 2 ratings
            self.popular = np.where(self.num_ratings > 2)[0]

            # define high-variance movies: movies that have recieved at least 5
            # ratings and have variance of rating values recieved of at least 2
            var_movies = count_movie_variability(self.ratings_df)
            self.high_var = []
            for i in np.where(self.num_ratings >= 5)[0]:
                if len(np.unique(var_movies[i:])) >= 2:
                    self.high_var.append(i)

        # load data in format that SVD can use
        reader = Reader(rating_scale=(1.0,5.0))
```

```
        self.data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']],
                                         reader=reader)
        self.k = k


    def trim_popular(self, testset):
        '''
        Trims testset so that only movies in popular and unpopular trimmed
        subsets movies remain

        Returns trimmed testset lists
        '''
        pop_test = []
        unpop_test = []
        for i in np.arange(len(testset)):
            for j in self.popular:
                # if movie is in the popular subset
                if j == testset[i][1]:
                    pop_test.append(testset[i])
                    break
                # since subset_movies is ordered from least to greatest,
                # if j > testset[i][1], then we know testset[i][1] is not in
                # popular set, so therefore it is in the unpopluar set
                elif j > testset[i][1]:
                    unpop_test.append(testset[i])
                    break
        return pop_test, unpop_test


    def trim_var(self, testset):
        '''
        Trims testset so that only movies in high variance trimmed movies subsets
        movies remain

        Returns trimmed testset lists
        '''
        var_test = []
        for i in np.arange(len(testset)):
            for j in np.arange(len(self.high_var)):
                # if movie is in the popular subset
                if j == testset[i][1]:
                    var_test.append(testset[i])
                    break
                # since subset_movies is ordered from least to greatest,
                # if j > testset[i][1], then we know testset[i][1] is not in
                # high variance data set
                elif j > testset[i][1]:
                    break
        return var_test


    def predict(self):
```

```
def predict(self):
    '''
    Performs 10-fold cv over different values of k for full data set

    Returns average RMSE and MAE for each value of k
    '''
    rmse = np.zeros(len(self.k))
    mae = np.zeros(len(self.k))
    for i in np.arange(len(k)):
        print(k[i])
        mf_bias = cross_validate(algo=SVD(n_factors=k[i], biased=True),
                        data=self.data, cv=10, n_jobs=-1)
        rmse[i] = mf_bias['test_rmse'].mean()
        mae[i] = mf_bias['test_mae'].mean()

    return rmse, mae


def predict_trim(self):
    '''
    Trains the MF w/ bias collaborative filter across 10-folds for different
    values of k and evaluates performance on popular, unpopular, and high
    variance validation sets

    Returns average RMSE across folds for all trimmed test sets
    '''

    rmse_pop = np.zeros(len(self.k))
    rmse_unpop = np.zeros(len(self.k))
    rmse_var = np.zeros(len(self.k))

    for i in np.arange(len(self.k)):
        print(k[i])

        # split data in 10 folds
        kf = KFold(n_splits=10)
        # define MF w/ bias collaborative filter
        mf_bias = SVD(n_factors=k[i], biased=True)

        total_RMSE_pop = 0
        total_RMSE_unpop = 0
        total_RMSE_var = 0
        for trainset, testset in kf.split(self.data):

            # train and test algorithm
            mf_bias.fit(trainset)
            pop_test, unpop_test = self.trim_popular(testset)
            var_test = self.trim_var(testset)
            pred_pop = mf_bias.test(pop_test)
            pred_unpop = mf_bias.test(unpop_test)
            pred_var = mf_bias.test(var_test)
```

```
                # compute RMSEs for fold
                total_RMSE_pop += accuracy.rmse(pred_pop, verbose=False)
                total_RMSE_unpop += accuracy.rmse(pred_unpop, verbose=False)
                total_RMSE_var += accuracy.rmse(pred_var, verbose=False)

            rmse_pop[i] = total_RMSE_pop / 10
            rmse_unpop[i] = total_RMSE_unpop / 10
            rmse_var[i] = total_RMSE_var / 10

            print("Popular RMSE: ", rmse_pop[i])
            print("Unpopular RMSE: ", rmse_unpop[i])
            print("High variation RMSE: ", rmse_var[i])
            print()

        return rmse_pop, rmse_unpop, rmse_var


def plot_rmse_mae(rmse, mae=None):
    '''
    Plots RMSE and MAE as a function of k
    '''

    if mae is not None:
        # Plot MAE as a function of k
        plt.plot(k, mae)
        plt.title("k versus MAE")
        plt.xlabel("k")
        plt.xticks(k)
        plt.ylabel("MAE")
        plt.show()

    # Plot RMSE as a function of k
    plt.plot(k, rmse)
    plt.title("k versus RMSE")
    plt.xlabel("k")
    plt.xticks(k)
    plt.ylabel("RSME")
    plt.show()


def report_lowest_RMSE(rmse, k):
    '''
    Returns minimum average RMSE value
    '''
    print("Minimum RMSE value: ", np.min(rmse))
    print("at k: ", k[np.argmin(rmse)])
    return np.min(rmse)

def report_lowest_MAE(mae, k):
    '''
     Returns minimum average MAE value
    '''
```

```
    print("Minimum MAE value: ", np.min(mae))
    print("at k: ", k[np.argmin(mae)])
    return np.min(mae)



def count_movie_ratings(ratings_df):
    '''
    Counts the number of times each movie was rated

    Returns an array where the index corresonds to the movieId and the value of
    the index corresponds to the number of times the movie was rated
    '''
    num_movies = np.max(np.unique(ratings_df['movieId']))
    num_ratings = np.zeros(num_movies)
    # loop through the rows of ratings.df and count the number of times each
    # movie was rated
    for i in np.arange(ratings_df.shape[0]):
        row_i = ratings_df.loc[i]
        movie_id = int(row_i[1]-1)
        num_ratings[movie_id] += 1
    return num_ratings


def count_movie_variability(ratings_df):
    '''
    Counts the variability in movie ratings

    Returns an array where the index corresponds to the movieId and the value of
    each column corresponds to the number of times the movie was given
    rating = (column # +1)
    '''
    num_movies = np.max(np.unique(ratings_df['movieId']))
    var_ratings = np.zeros((num_movies, 5))
    # loop through the rows of ratings.df and count the number of times each
    # movie was given a certain ratings
    for i in np.arange(ratings_df.shape[0]):
        row_i = ratings_df.loc[i]
        movie_id = int(row_i[1]-1)
        rating = int(row_i[2])
        var_ratings[movie_id, rating-1] += 1
    return var_ratings
```

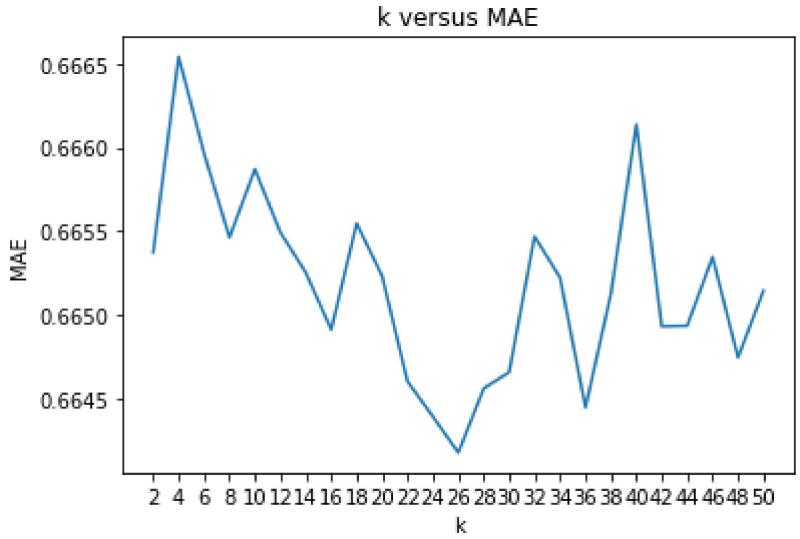
Question 24-25

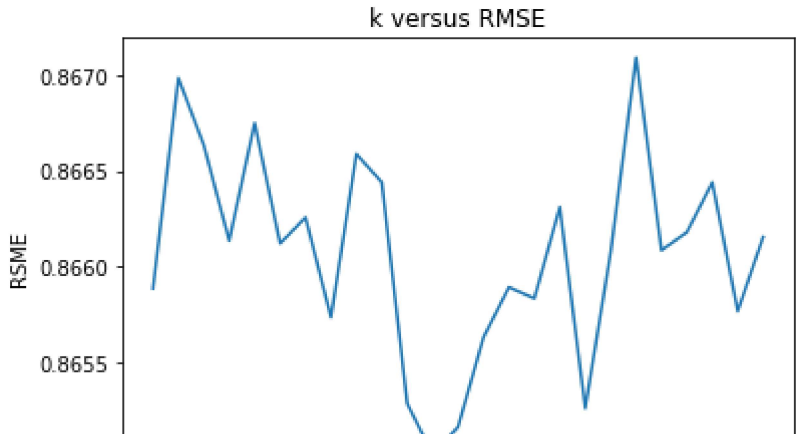
```
'''
Perform 10-fold cross-validation using a MF w/ bias collaborative filter
'''
```

```
ratings_df = pd.read_csv('/content/drive/MyDrive/ECE_219/3. Project 3 - Recommendation System

# compute RMSE and MAE as a function of k, and plot results
k = np.arange(2, 51, 2)
mfb = MF_Bias(ratings_df, k)
rmse, mae = mfb.predict()
```

```
     2
     4
     6
     8
     10
     12
     14
     16
     18
     20
     22
     24
     26
     28
     30
     32
     34
     36
     38
     40
     42
     44
     46
     48
     50
```

```
'''
Plot RMSE and MAE as a function of k and report the minimum RMSE and MAE
'''

plot_rmse_mae(rmse, mae)
print()
print()
print("Minimum avg RMSE: ", report_lowest_RMSE(rmse, k))
print("Minimum avg MAE: ", report_lowest_MAE(mae, k))
```





Question 26-28

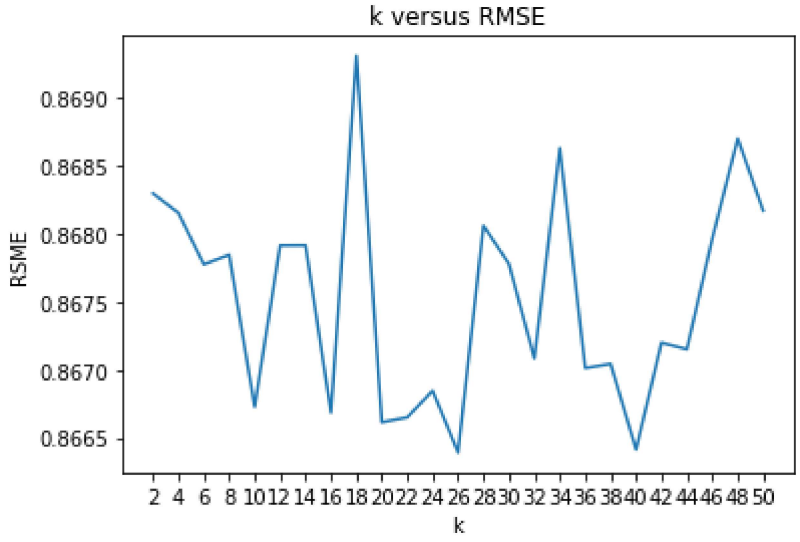
```
'''
Perform 10-fold cross-validation using a MF w/ bias collaborative filer on the
populur, unpopular, and high variance movie subsets
'''

# load data
ratings_df = pd.read_csv('/content/drive/MyDrive/ECE_219/3. Project 3 - Recommendation System

# compute RMSE as a function of k for each trimmed data set
k = np.arange(2, 51, 2)
mfb = MF_Bias(ratings_df, k, trim=True)
rmse_pop, rmse_unpop, rmse_var = mfb.predict_trim()
```

```
2
Popular RMSE:  0.8682986955207219
Unpopular RMSE:  0.8648462128000152
High variation RMSE:  0.8739938456371131

4
Popular RMSE:  0.8681547070701848
Unpopular RMSE:  0.8655502835275273
```

```
        High variation RMSE:  0.8745270212971107

        6
        Popular RMSE:  0.8677755927608825
        Unpopular RMSE:  0.8650596318710637
        High variation RMSE:  0.8735927529851354

        8
        Popular RMSE:  0.8678464203150845
        Unpopular RMSE:  0.8641511079191042
        High variation RMSE:  0.8734010715497433

        10
        Popular RMSE:  0.8667283741261075
        Unpopular RMSE:  0.8650631468365224
        High variation RMSE:  0.872237895430802

        12
        Popular RMSE:  0.8679154835546564
        Unpopular RMSE:  0.8647329684886481
        High variation RMSE:  0.8729428584156448

        14
        Popular RMSE:  0.8679161236574672
        Unpopular RMSE:  0.864201441096634
        High variation RMSE:  0.8733339984549258

        16
        Popular RMSE:  0.8666855257976479
        Unpopular RMSE:  0.8642546545304484
        High variation RMSE:  0.8722075077955205

        18
        Popular RMSE:  0.869311408784163
        Unpopular RMSE:  0.8655785410146197
        High variation RMSE:  0.8742353784714914

        20
        Popular RMSE:  0.8666140265649471
        Unpopular RMSE:  0.8641087267760887
        High variation RMSE:  0.8718387170165782

        22
        Popular RMSE:  0.8666501581290902
        Unpopular RMSE:  0.8639752181683799
        High variation RMSE:  0.8724996712177451

        24
        Popular RMSE:  0.8668462251027187
        Unpopular RMSE:  0.8643553479013084
        High variation RMSE:  0.8726338070429506


    print("RMSE for Popular Movies")
    plot_rmse_mae(rmse_pop)
    print()
    print()
    print("Minimum avg RMSE: ", report_lowest_RMSE(rmse_pop, k))
```

```
print("Minimum avg RMSE: ", report_lowest_RMSE(rmse_pop, k))
```

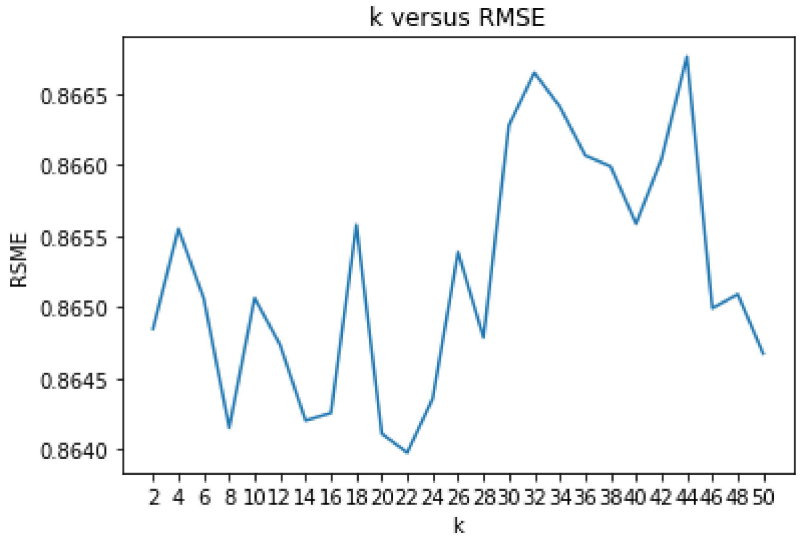
```
RMSE for Popular Movies
```




```
Lowest RMSE value:  0.8663915719248184
at k:  26
Minimum avg RMSE:  0.8663915719248184
```

```
print("RMSE for Unpopular Movies")
plot_rmse_mae(rmse_unpop)
print()
print()
print("Minimum avg RMSE: ", report_lowest_RMSE(rmse_unpop, k))
```

```
RMSE for Unpopular Movies
```

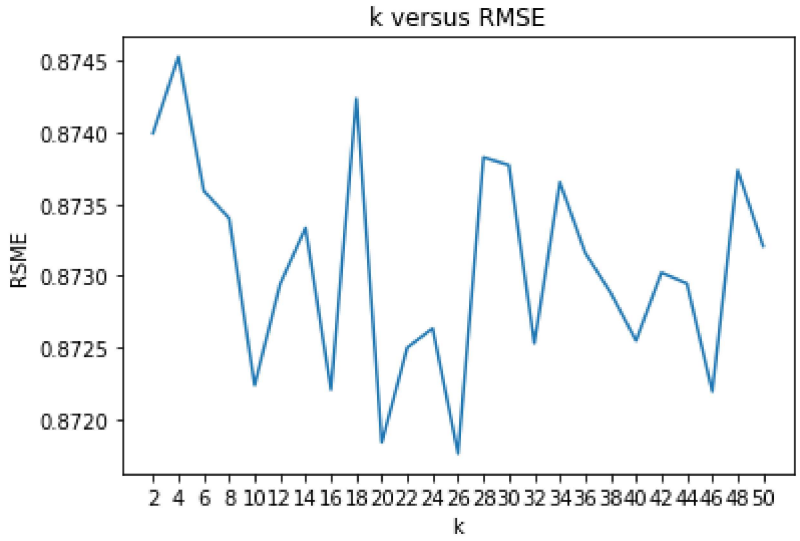



```
Lowest RMSE value:  0.8639752181683799
at k:  22
Minimum avg RMSE:  0.8639752181683799
```

```
print("RMSE for High Variance Movies")
plot_rmse_mae(rmse_var)
```

```
plot_rmse_mae(rmse_var)
print()
print()
print("Minimum avg RMSE: ", report_lowest_RMSE(rmse_var, k))
```

```
RMSE for High Variance Movies
```

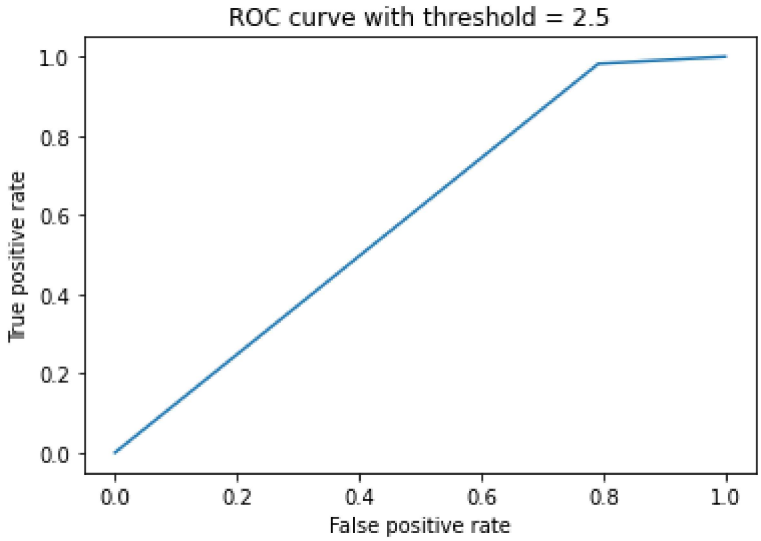



```
Lowest RMSE value:  0.871761886678672
at k:  26
Minimum avg RMSE:  0.871761886678672
```

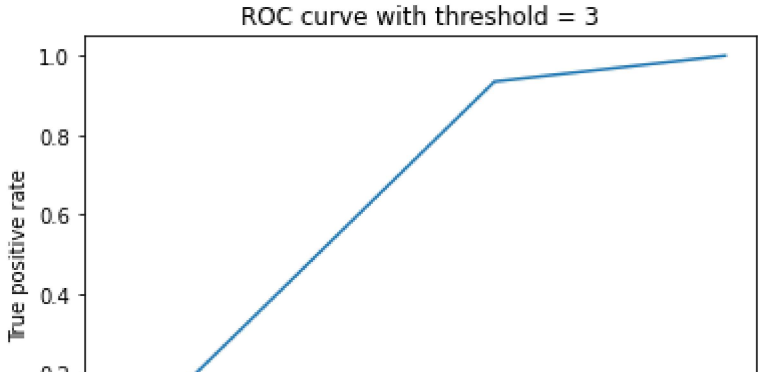
Question 29

```
'''
Plot ROC curves for various threshold values and report AUC values
'''

from surprise import Reader
from surprise import Dataset
from surprise.model_selection.split import train_test_split
from surprise.prediction_algorithms.matrix_factorization import SVD
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt


# load data in format that SVD can use
ratings_df = pd.read_csv('/content/drive/MyDrive/ECE_219/3. Project 3 - Recommendation Systen
reader = Reader(rating_scale=(1.0,5.0))
data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader=reader)
trainset, testset = train_test_split(data, test_size=0.1)

# define MF w/ bias collaborative filter
k = 26
```

```
    mf_bias = SVD(n_factors=k, biased=True)

    # train filter and predict
    mf_bias.fit(trainset)
    preds = mf_bias.test(testset)

    # print ROC curve for different thresholds
    binary_thresholds = [2.5, 3, 3.5, 4]
    for t in binary_thresholds:
        y_true = []
        y_pred = []
        for i in np.arange(len(preds)):
            y_pred.append(1 if preds[i].est > t else 0)
            y_true.append(1 if preds[i].r_ui > t else 0)

        fpr, tpr, _ = roc_curve(y_true, y_pred)
        auc_ = roc_auc_score(y_true,  y_pred)

        plt.plot(fpr,tpr)
        plt.xlabel('False positive rate')
        plt.ylabel('True positive rate')
        plt.title('ROC curve with threshold = {}'.format(t))
        plt.show()
        print('The area under the curve (AUC) value is {}'.format(auc_))
```




```
The area under the curve (AUC) value is 0.5949492560397334
```



Naive Collaborative Filtering

False positive rate

```
'''
Functions for Questions 30 - 33
'''
from surprise import Reader
from surprise import Dataset
from surprise.model_selection import KFold
from surprise import accuracy
import matplotlib.pyplot as plt


class Naive:
    '''
    Class to design an 10-fold cross validation using a naive collaborative
    filter
    '''

    def __init__(self, ratings_df, trim=False):
        '''
        data: data this is performed on; a DataFrame
        k: a list storing the values of k
        '''
```

```
        self.ratings_df = ratings_df

        if trim==True:
            # count number of ratings for each movie
            self.num_ratings = count_movie_ratings(self.ratings_df)

            # define popular movies: movies that have recieved more than 2 ratings
            self.popular = np.where(self.num_ratings > 2)[0]

            # define high-variance movies: movies that have recieved at least 5
            # ratings and have variance of rating values recieved of at least 2
            var_movies = count_movie_variability(self.ratings_df)
            self.high_var = []
            for i in np.where(self.num_ratings >= 5)[0]:
                if len(np.unique(var_movies[i:])) >= 2:
                    self.high_var.append(i)


        # count number of ratings for each user
        self.user_ratings = count_user_ratings(self.ratings_df)

        # compute average rating for each user, (mu_i)
        self.avg_ratings = avg_user_rating(self.user_ratings)

        # load data in format that SVD can use
        reader = Reader(rating_scale=(1.0,5.0))
        self.data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']],
                                         reader=reader)


    def trim_popular(self, testset):
        '''
        Trims testset so that only movies in popular and unpopular trimmed
        subsets movies remain

        Returns trimmed testset lists
        '''
        pop_test = []
        unpop_test = []
        for i in np.arange(len(testset)):
            for j in self.popular:
                # if movie is in the popular subset
                if j == testset[i][1]:
                    pop_test.append(testset[i])
                    break
                # since subset_movies is ordered from least to greatest,
                # if j > testset[i][1], then we know testset[i][1] is not in
                # popular set, so therefore it is in the unpopluar set
                elif j > testset[i][1]:
                    unpop_test.append(testset[i])
                    break
```

```
        return pop_test, unpop_test

    def trim_var(self, testset):
        '''
        Trims testset so that only movies in high variance trimmed movies subsets
        movies remain

        Returns trimmed testset lists
        '''
        var_test = []
        for i in np.arange(len(testset)):
            for j in np.arange(len(self.high_var)):
                # if movie is in the popular subset
                if j == testset[i][1]:
                    var_test.append(testset[i])
                    break
                # since subset_movies is ordered from least to greatest,
                # if j > testset[i][1], then we know testset[i][1] is not in
                # high variance data set
                elif j > testset[i][1]:
                    break
        return var_test


    def find_fold_error(self, testset):
        '''
        Determine error for one fold of k-fold cross validation

        Returns error for that fold
        '''
        fold_error = 0
        for i in np.arange(len(testset)):
            # find userId in test set
            userId = testset[i][0]-1
            # find avg move rating for user
            pred_rating = self.avg_ratings[userId]
            # compute error and add to running error total
            fold_error += (testset[i][2] - pred_rating)**2
        return fold_error


    def predict(self):
        '''
        Trains the naive collaborative filter across 10-fold and evaluates
        performance on validation set

        Returns: avg RMSE for all folds
        '''
        kf = KFold(n_splits=10)
        total_RMSE=0
        for trainset, testset in kf.split(self.data):
            fold_error = self.find_fold_error(testset)
```

```
            total_RMSE += np.sqrt(fold_error / len(testset))
        avg_RMSE = total_RMSE / 10
        return avg_RMSE


    def predict_on_trim(self):
        '''
        Trains the naive collaborative filter across 10-fold and evaluates
        performance on validation set (a subset of movies)

        Returns: avg RMSE for all folds for each trimmed set
        '''
        total_RMSE_pop = 0
        total_RMSE_unpop = 0
        total_RMSE_var = 0

        kf = KFold(n_splits=10)
        for trainset, testset in kf.split(self.data):
            # trim testset
            pop_test, unpop_test = self.trim_popular(testset)
            var_test = self.trim_var(testset)
            # calculate error for fold
            fold_error_pop = self.find_fold_error(pop_test)
            fold_error_unpop = self.find_fold_error(unpop_test)
            fold_error_var = self.find_fold_error(var_test)

            # add fold RMSE to total RMSE
            print("Popular RMSE:, ", np.sqrt(fold_error_pop / len(pop_test)))
            total_RMSE_pop += np.sqrt(fold_error_pop / len(pop_test))
            print("Unpopular RMSE:, ", np.sqrt(fold_error_unpop / len(unpop_test)))
            total_RMSE_unpop += np.sqrt(fold_error_unpop / len(unpop_test))
            print("High variance RMSE:, ", np.sqrt(fold_error_var / len(var_test)))
            total_RMSE_var += np.sqrt(fold_error_var / len(var_test))
            print()

        avg_RMSE_pop = total_RMSE_pop / 10
        avg_RMSE_unpop = total_RMSE_unpop / 10
        avg_RMSE_var = total_RMSE_var / 10
        return avg_RMSE_pop, avg_RMSE_unpop, avg_RMSE_var



def count_movie_ratings(ratings_df):
    '''
    Counts the number of times each movie was rated

    Returns an array where the index corresonds to the movieId and the value of
    the index corresponds to the number of times the movie was rated
    '''
    num_movies = np.max(np.unique(ratings_df['movieId']))
    num_ratings = np.zeros(num_movies)
    # loop through the rows of ratings_df and count the number of times each
```

```
        # loop through the rows of ratings.df and count the number of times each
        # movie was rated
        for i in np.arange(ratings_df.shape[0]):
            row_i = ratings_df.loc[i]
            movie_id = int(row_i[1]-1)
            num_ratings[movie_id] += 1
        return num_ratings

    def count_movie_variability(ratings_df):
        '''
        Counts the variability in movie ratings

        Returns an array where the index corresponds to the movieId and the value of
        each column corresponds to the number of times the movie was given
        rating = (column # +1)
        '''
        num_movies = np.max(np.unique(ratings_df['movieId']))
        var_ratings = np.zeros((num_movies, 5))
        # loop through the rows of ratings.df and count the number of times each
        # movie was given a certain ratings
        for i in np.arange(ratings_df.shape[0]):
            row_i = ratings_df.loc[i]
            movie_id = int(row_i[1]-1)
            rating = int(row_i[2])
            var_ratings[movie_id, rating-1] += 1
        return var_ratings


    def count_user_ratings(ratings_df):
        '''
        Counts the number of times each user rated a movie

        Returns an array where the row index corresponds to userId, and the values
        in the columns correspond to the number of movies the person gave that
        rating to
        '''
        num_users = np.max(np.unique(ratings_df['userId']))
        user_ratings = np.zeros((num_users, 5))
        for i in np.arange(ratings_df.shape[0]):
            row_i = ratings_df.loc[i]
            user_id = int(row_i[0]-1)
            rating = int(row_i[2])
            user_ratings[user_id, rating-1] += 1
        return user_ratings

    def avg_user_rating(user_ratings):
        '''
        Counts avg movie rating for each user

        Returns an array where the index number corresponds to userId and the value
        of the index corresponds to the average rating for that given user
        '''
```

```
    avg_ratings = np.zeros(user_ratings.shape[0])
    for n in np.arange(user_ratings.shape[0]):
        row_n = user_ratings[n,:]
        num_movies_rated = np.sum(row_n)
        total_ratings = 0
        for i in np.arange(len(row_n)):
            total_ratings += (i+1) * row_n[i]

        avg_ratings[n] = total_ratings / num_movies_rated
    return avg_ratings
```

Question 30

```
'''
Perform 10-fold cross-validation using a naice collaborative filter
'''

from surprise import Reader
from surprise import Dataset
from surprise.model_selection import KFold


# load data
ratings_df = pd.read_csv('/content/drive/MyDrive/ECE_219/3. Project 3 - Recommendation System

# define naive filter and get RMSE of predictions
naive = Naive(ratings_df)
avg_RMSE = naive.predict()
print("avg RMSE: ", avg_RMSE)
```

```
 avg RMSE:  0.9523262510567827
```

Question 31 - 33

```
from surprise import Reader
from surprise import Dataset
from surprise.model_selection import KFold


# load data
ratings_df = pd.read_csv('/content/drive/MyDrive/ECE_219/3. Project 3 - Recommendation System

# define naive filter and get RMSE of predictions
naive = Naive(ratings_df, trim=True)
avg_RMSE_pop, avg_RMSE_unpop, avg_RMSE_var = naive.predict_on_trim()
print("avg Popular RMSE: ", avg_RMSE_pop)
```

```
print("avg Popular RMSE: ", avg_RMSE_pop)
print()
print("avg Unpopular RMSE: ", avg_RMSE_unpop)
print()
print("avg High Variance RMSE: ", avg_RMSE_var)
```

```
Popular RMSE:,  0.9825866975143619
Unpopular RMSE:,  0.9321266225301313
High variance RMSE:,  0.9825921297776363

Popular RMSE:,  0.9526596342166747
Unpopular RMSE:,  0.9417379909679282
High variance RMSE:,  0.9679738788912152

Popular RMSE:,  0.9793705049921972
Unpopular RMSE:,  0.9435409073938429
High variance RMSE:,  0.9784851721176717

Popular RMSE:,  0.9547556197721074
Unpopular RMSE:,  0.9527948469846832
High variance RMSE:,  0.9646345194200495

Popular RMSE:,  0.9398641673886649
Unpopular RMSE:,  0.9486819001662301
High variance RMSE:,  0.9592063012096851

Popular RMSE:,  0.9612202924520884
Unpopular RMSE:,  0.9515918439864646
High variance RMSE:,  0.9734623881427328

Popular RMSE:,  0.9671545470105216
Unpopular RMSE:,  0.9390745697592052
High variance RMSE:,  0.9680193056460937

Popular RMSE:,  0.96309936930674
Unpopular RMSE:,  0.9215409793417684
High variance RMSE:,  0.9703616734107211

Popular RMSE:,  0.9558445291476694
Unpopular RMSE:,  0.9363951465936508
High variance RMSE:,  0.9745761618114263

Popular RMSE:,  0.958914352444468
Unpopular RMSE:,  0.9588270597444347
High variance RMSE:,  0.964659370977253

avg Popular RMSE:  0.9615469714245493

avg Unpopular RMSE:  0.9426311867468338

avg High Variance RMSE:  0.9703970901404485
```

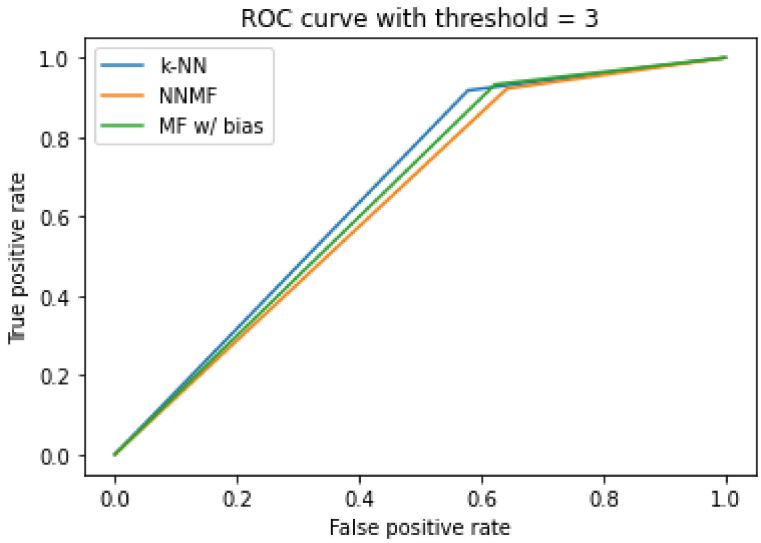
Performance Comparison

Question 34

```
'''
Plot ROC curves for threshold=3 for each of k-NN, NNMF, and MF w/ bias to
compare their performance
'''

from surprise import Reader
from surprise import Dataset
from surprise.model_selection.split import train_test_split
from surprise.prediction_algorithms.knns import KNNWithMeans
from surprise.prediction_algorithms.matrix_factorization import NMF
from surprise.prediction_algorithms.matrix_factorization import SVD
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt


# load data in format that SVD can use
ratings_df = pd.read_csv('/content/drive/MyDrive/ECE_219/3. Project 3 - Recommendation System
reader = Reader(rating_scale=(1.0,5.0))
data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader=reader)
trainset, testset = train_test_split(data, test_size=0.1)

# train k-NN collaborative filter, get TPR and FPR
knn = KNNWithMeans(k=14, sim_options={'name': 'pearson'})
knn.fit(trainset)
preds = knn.test(testset)
y_true = []
y_pred = []
for i in np.arange(len(preds)):
    y_pred.append(1 if preds[i].est > 3 else 0)
    y_true.append(1 if preds[i].r_ui > 3 else 0)
knn_fpr, knn_tpr, _ = roc_curve(y_true, y_pred)

# train NNMF collaborative filter, get TPR and FPR
nnmf = NMF(n_factors=20, biased=False)
nnmf.fit(trainset)
preds = nnmf.test(testset)
y_true = []
y_pred = []
for i in np.arange(len(preds)):
    y_pred.append(1 if preds[i].est > 3 else 0)
    y_true.append(1 if preds[i].r_ui > 3 else 0)
nnmf_fpr, nnmf_tpr, _ = roc_curve(y_true, y_pred)

# train MF w/ bias collaborative filter, get TPR and FPR
mf_bias = SVD(n_factors=26, biased=True)
mf_bias.fit(trainset)
```

```
preds = mf_bias.test(testset)
y_true = []
y_pred = []
for i in np.arange(len(preds)):
    y_pred.append(1 if preds[i].est > 3 else 0)
    y_true.append(1 if preds[i].r_ui > 3 else 0)
mf_fpr, mf_tpr, _ = roc_curve(y_true, y_pred)


# print ROC curve
plt.plot(knn_fpr, knn_tpr, label='k-NN')
plt.plot(nnmf_fpr, nnmf_tpr, label='NNMF')
plt.plot(mf_fpr, mf_tpr, label='MF w/ bias')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve with threshold = 3')
plt.legend()
plt.show()
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
```



Ranking

Question 36

```
'''
QUESTION 36: Plot average precision (Y-axis) against t (X-axis) for the ranking obtained
using k-NN collaborative filter predictions. Also, plot the average recall (Y-axis) against t
and average precision (Y-axis) against average recall (X-axis).
'''


from collections import defaultdict
```

```
from collections import defaultdict
from surprise.model_selection.validation import cross_validate
from surprise.prediction_algorithms.knns import KNNWithMeans
from surprise import Dataset
from surprise import Reader
from surprise import accuracy
from surprise.model_selection import KFold
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pdb
from surprise import SVD
from surprise import Dataset
import time

# code from surprise: https://surprise.readthedocs.io/en/stable/FAQ.html



def precision_recall_at_k_for_t_items(predictions, t, threshold=3):
    # First map the predictions to each user.
    user_est_true = defaultdict(list)
    for uid, _, true_r, est, _ in predictions:
        user_est_true[uid].append((est, true_r))

    precisions = dict()
    recalls = dict()
    for uid, user_ratings in user_est_true.items():
        # Sort user ratings by estimated value
        user_ratings.sort(key=lambda x: x[0], reverse=True)

        # Number of recommended items is t
        n_rec_k = t

        # Number of relevant and recommended items in top t
        n_rel_and_rec_k = 0
        for i in range(t):
            if user_ratings[i][1] >= threshold:
                n_rel_and_rec_k += 1

        # Number of relevant items
        n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)

        # Precision@K: Proportion of recommended items that are relevant
        precisions[uid] = n_rel_and_rec_k / n_rec_k

        # Recall@K: Proportion of relevant items that are recommended
        recalls[uid] = n_rel_and_rec_k / n_rel

    return precisions, recalls
```

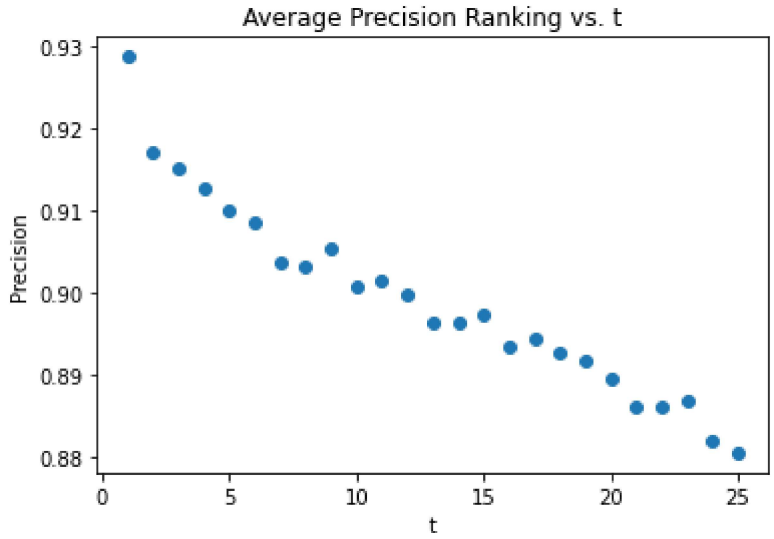
```
def top_t_items(testset, t, threshold=3):
    final = []
    dict1 = {}
    dict2 = {}

    for (x, y, z) in testset: #(x= uid, y = moviedid, rating)
        if x not in  dict1:
             dict1[x] = 0 #if uid not in dict1 append
        dict1[x] += 1
        if x not in  dict2: #if uid is not in dict2 then append
            dict2[x] = 0
        if z >= threshold: # if movie rating is greater than our threshold
            dict2[x] += 1 #increase the value of uid

    for (x, y, z) in testset:
        if dict1[x] >= t and dict2[x] > 0: #if the uid is greater
          final.append((x, y, z )) #than our threshold and greater than
    return final #return a valid movieid,userid, rating tuple



ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
reader = Reader(rating_scale=(0.5, 5.0))
data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader)

t1= time.time()

k_folds = KFold(10)
threshold = 3
#values = list(range(1, 26))
values = np.arange(1, 26)
results = np.zeros((2, 26))



for t in values:
        recall_values = []
        precision_values = []
        for trainset, testset in k_folds.split(data):

            algo = KNNWithMeans(k=14, sim_options={'name': 'pearson'})
            #as usual fit on trainset
            algo.fit(trainset)

            testset_ = top_t_items(testset, t, threshold=3)
            predictions = algo.test(testset_)
            #return predictions on the testset for which the conditions for t and g are met

            precisions, recalls = precision_recall_at_k_for_t_items(predictions, t, threshold
            #average over the fold
            precision_mean = sum(prec for prec in precisions.values()) / len(precisions)
```

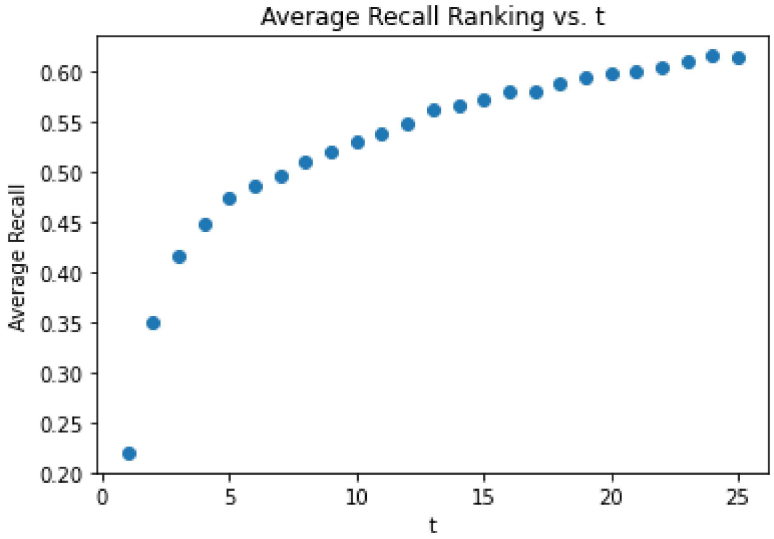
```
            precision_mean = sum(prec for prec in precisions.values()) / len(precisions)
            recall_mean = sum(rec for rec in recalls.values()) / len(recalls)
            #store
            precision_values.append(precision_mean)
            recall_values.append(recall_mean)
        #average
        results[1,t]= np.mean(precision_values) #results 1 is precision
        results[0,t] = np.mean(recall_values)  #results 0 is recall



final_time = time.time() - t1
k_final_values= np.copy(results)
print('This took {} seconds'.format(final_time))
```

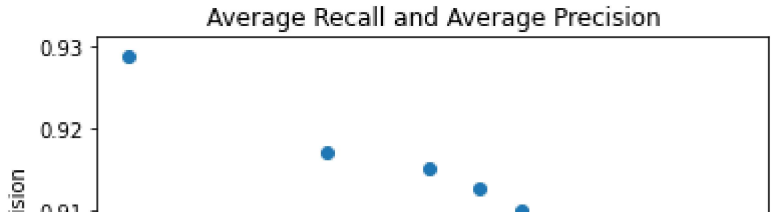
```
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
```

```
plt.title('Average Precision Ranking vs. t')
plt.xlabel('t')
plt.ylabel('Average Precision')
plt.ylabel('Precision')
plt.scatter(values,results[1, 1:26])
plt.show()

plt.title('Average Recall Ranking vs. t')
plt.xlabel('t')
plt.ylabel('Average Recall')
plt.scatter(values, results[0, 1:26])
plt.show()

precision_final = results[1, 1:26]
recall_final = results[0, 1:26]

plt.title('Average Recall and Average Precision')
plt.xlabel('Average Recall')
plt.ylabel('Average Precision')
plt.scatter(results[0, 1:26], results[1, 1:26])
plt.show()
```







Question 37

```
'''
QUESTION 37: Plot average precision (Y-axis) against t (X-axis) for the ranking obtained
using NMF collaborative filter predictions. Also, plot the average recall (Y-axis) against t
and average precision (Y-axis) against average recall (X-axis).
'''

from collections import defaultdict
from surprise.model_selection.validation import cross_validate
from surprise.prediction_algorithms.matrix_factorization import NMF
from surprise import Dataset
from surprise import Reader
from surprise import accuracy
```

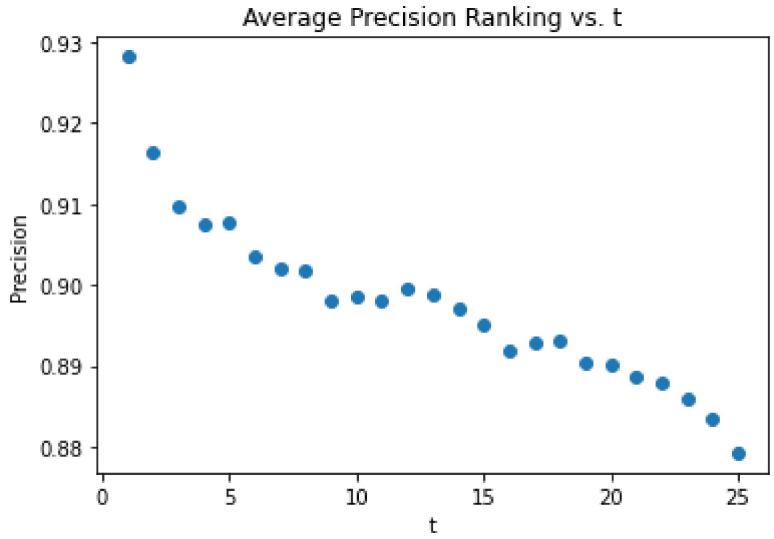
```
from surprise.model_selection import KFold
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pdb
from surprise import SVD
from surprise import Dataset
import time

# code from surprise: https://surprise.readthedocs.io/en/stable/FAQ.html



def precision_recall_at_k_for_t_items(predictions, t, threshold=3):
    # First map the predictions to each user.
    user_est_true = defaultdict(list)
    for uid, _, true_r, est, _ in predictions:
        user_est_true[uid].append((est, true_r))

    precisions = dict()
    recalls = dict()
    for uid, user_ratings in user_est_true.items():
        # Sort user ratings by estimated value
        user_ratings.sort(key=lambda x: x[0], reverse=True)

        # Number of recommended items is t
        n_rec_k = t

        # Number of relevant and recommended items in top t
        n_rel_and_rec_k = 0
        for i in range(t):
            if user_ratings[i][1] >= threshold:
                n_rel_and_rec_k += 1

        # Number of relevant items
        n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)

        # Precision@K: Proportion of recommended items that are relevant
        precisions[uid] = n_rel_and_rec_k / n_rec_k

        # Recall@K: Proportion of relevant items that are recommended
        recalls[uid] = n_rel_and_rec_k / n_rel

    return precisions, recalls



def top_t_items(testset, t, threshold=3):
    final = []
    dict1 = {}
    dict2 = {}
```

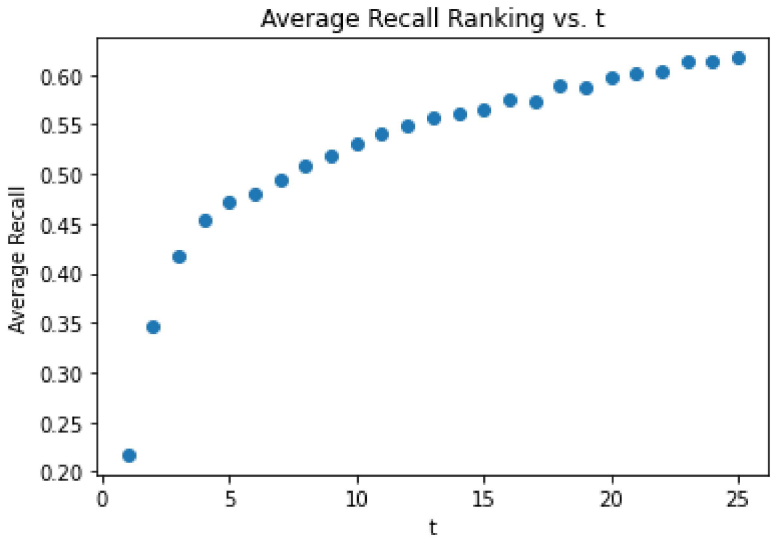
```
    for (x, y, z) in testset: #(x= uid, y = moviedid, rating)
        if x not in  dict1:
             dict1[x] = 0 #if uid not in dict1 append
        dict1[x] += 1
        if x not in  dict2: #if uid is not in dict2 then append
            dict2[x] = 0
        if z >= threshold: # if movie rating is greater than our threshold
            dict2[x] += 1 #increase the value of uid

    for (x, y, z) in testset:
        if dict1[x] >= t and dict2[x] > 0: #if the uid is greater
          final.append((x, y, z )) #than our threshold and greater than
    return final #return a valid movieid,userid, rating tuple



ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
reader = Reader(rating_scale=(0.5, 5.0))
data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader)

t1= time.time()

k_folds = KFold(10)
threshold = 3
#values = list(range(1, 26))
values = np.arange(1, 26)
results = np.zeros((2, 26))



for t in values:
        recall_values = []
        precision_values = []
        for trainset, testset in k_folds.split(data):

            algo = NMF(n_factors = 20, biased = False)
            #as usual fit on trainset
            algo.fit(trainset)

            testset_ = top_t_items(testset, t, threshold=3)
            predictions = algo.test(testset_)
            #return predictions on the testset for which the conditions for t and g are met

            precisions, recalls = precision_recall_at_k_for_t_items(predictions, t, threshold
            #average over the fold
            precision_mean = sum(prec for prec in precisions.values()) / len(precisions)
            recall_mean = sum(rec for rec in recalls.values()) / len(recalls)
            #store
            precision_values.append(precision_mean)
            recall_values.append(recall_mean)
        #average
```

```
            results[1,t]= np.mean(precision_values) #results 1 is precision
            results[0,t] = np.mean(recall_values)  #results 0 is recall


    nmf_final_values = np.copy(results)
    final_time = time.time() - t1

    print('This took {} seconds'.format(final_time))
```

```
This took 1989.286387681961 seconds
```

```
    plt.title('Average Precision Ranking vs. t')
    plt.xlabel('t')
    plt.ylabel('Average Precision')
    plt.ylabel('Precision')
    plt.scatter(values,results[1, 1:26])
    plt.show()

    plt.title('Average Recall Ranking vs. t')
    plt.xlabel('t')
    plt.ylabel('Average Recall')
    plt.scatter(values, results[0, 1:26])
    plt.show()

    precision_final = results[1, 1:26]
    recall_final = results[0, 1:26]

    plt.title('Average Recall and Average Precision')
    plt.xlabel('Average Recall')
    plt.ylabel('Average Precision')
    plt.scatter(results[0, 1:26], results[1, 1:26])
    plt.show()
```





Question 38

```
'''
Plot average precision (Y-axis) against t (X-axis) for the ranking obtained
using MF w/ bias collaborative filter predictions. Also, plot the average recall
(Y-axis) against t (X-axis) and average precision (Y-axis) against average
recall (X-axis).
'''

from collections import defaultdict
from surprise.model_selection.validation import cross_validate
from surprise.prediction_algorithms.matrix_factorization import SVD
from surprise import Dataset
from surprise import Reader
from surprise import accuracy
from surprise.model_selection import KFold
import matplotlib.pyplot as plt
import pdb
from surprise import SVD
```

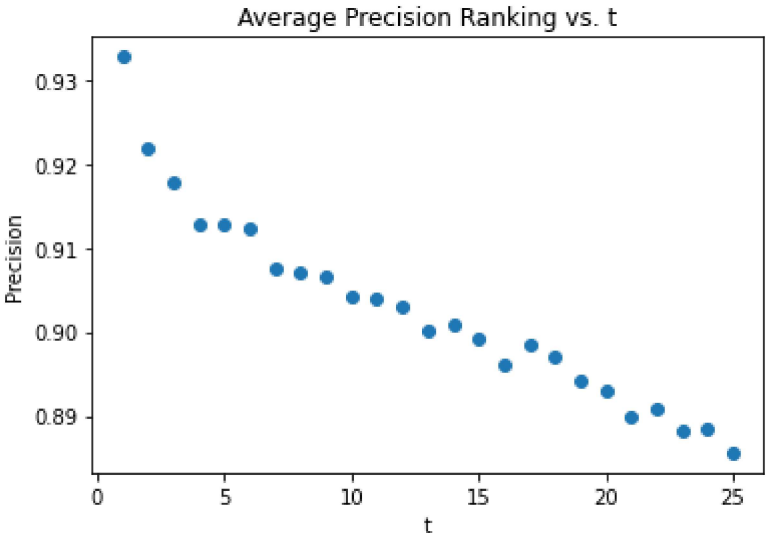
```
import time

# code from surprise: https://surprise.readthedocs.io/en/stable/FAQ.html



def precision_recall_at_k_for_t_items(predictions, t, threshold=3):
    # First map the predictions to each user.
    user_est_true = defaultdict(list)
    for uid, _, true_r, est, _ in predictions:
        user_est_true[uid].append((est, true_r))

    precisions = dict()
    recalls = dict()
    for uid, user_ratings in user_est_true.items():
        # Sort user ratings by estimated value
        user_ratings.sort(key=lambda x: x[0], reverse=True)

        # Number of recommended items is t
        n_rec_k = t

        # Number of relevant and recommended items in top t
        n_rel_and_rec_k = 0
        for i in range(t):
            if user_ratings[i][1] >= threshold:
                n_rel_and_rec_k += 1

        # Number of relevant items
        n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)

        # Precision@K: Proportion of recommended items that are relevant
        precisions[uid] = n_rel_and_rec_k / n_rec_k

        # Recall@K: Proportion of relevant items that are recommended
        recalls[uid] = n_rel_and_rec_k / n_rel

    return precisions, recalls



def top_t_items(testset, t, threshold=3):
    final = []
    dict1 = {}
    dict2 = {}

    for (x, y, z) in testset: #(x= uid, y = moviedid, rating)
        if x not in  dict1:
             dict1[x] = 0 #if uid not in dict1 append
        dict1[x] += 1
        if x not in  dict2: #if uid is not in dict2 then append
            dict2[x] = 0
        if z >= threshold: # if movie rating is greater than our threshold
```

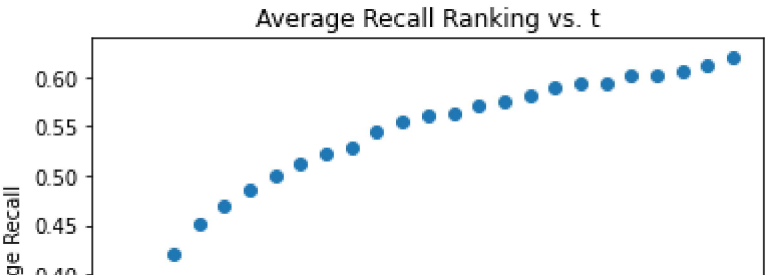
```
            dict2[x] += 1 #increase the value of uid

    for (x, y, z) in testset:
        if dict1[x] >= t and dict2[x] > 0: #if the uid is greater
          final.append((x, y, z )) #than our threshold and greater than
    return final #return a valid movieid,userid, rating tuple



ratings_df = pd.read_csv('/content/drive/MyDrive/ratings.csv')
reader = Reader(rating_scale=(0.5, 5.0))
data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader)

t1= time.time()

k_folds = KFold(10)
threshold = 3
#values = list(range(1, 26))
values = np.arange(1, 26)
results = np.zeros((2, 26))



for t in values:
        recall_values = []
        precision_values = []
        for trainset, testset in k_folds.split(data):

            algo = SVD(n_factors=26, biased=True)
            #as usual fit on trainset
            algo.fit(trainset)

            testset_ = top_t_items(testset, t, threshold=3)
            predictions = algo.test(testset_)
            #return predictions on the testset for which the conditions for t and g are met

            precisions, recalls = precision_recall_at_k_for_t_items(predictions, t, threshold
            #average over the fold
            precision_mean = sum(prec for prec in precisions.values()) / len(precisions)
            recall_mean = sum(rec for rec in recalls.values()) / len(recalls)
            #store
            precision_values.append(precision_mean)
            recall_values.append(recall_mean)
        #average
        results[1,t]= np.mean(precision_values) #results 1 is precision
        results[0,t] = np.mean(recall_values)  #results 0 is recall



final_time = time.time() - t1
mfwb_final_values = np.copy(results)
print('This took {} seconds'.format(final_time))
```

```
print('This took {} seconds'.format(final_time))
```

```
This took 748.2185685634613 seconds
```

```
plt.title('Average Precision Ranking vs. t')
plt.xlabel('t')
plt.ylabel('Average Precision')
plt.ylabel('Precision')
plt.scatter(values,results[1, 1:26])
plt.show()

plt.title('Average Recall Ranking vs. t')
plt.xlabel('t')
plt.ylabel('Average Recall')
plt.scatter(values, results[0, 1:26])
plt.show()

precision_final = results[1, 1:26]
recall_final = results[0, 1:26]

plt.title('Average Recall and Average Precision')
plt.xlabel('Average Recall')
plt.ylabel('Average Precision')
plt.scatter(results[0, 1:26], results[1, 1:26])
plt.show()
```





Question 39

```
print(k_final_values)
names= ['K-NN','NNMF', 'MF with bias']
print(nmf_final_values)
colors=['r','g','b']

plt.title('Average Recall and Average Precision')
plt.xlabel('Average Recall')
plt.ylabel('Average Precision')
plt.scatter(k_final_values[0, 1:26], k_final_values[1, 1:26], label = 'K-NN', color= colors[0
plt.scatter(nmf_final_values[0, 1:26], nmf_final_values[1, 1:26], label = names[1], color= co
plt.scatter(mfwb_final_values[0, 1:26], mfwb_final_values[1, 1:26], label = names[2], color=
plt.legend(names)
plt.show()
```

```
[[0.         0.21799843 0.34535581 0.41638207 0.45170841 0.47042043
  0.48128616 0.49910949 0.50596762 0.52230887 0.53077094 0.5440782
  0.54900693 0.56238721 0.56113431 0.56747563 0.57732803 0.57730336
  0.58499124 0.58784918 0.59532034 0.60198906 0.60346528 0.61383629
  0.61449064 0.61914797]
 [0.         0.92815617 0.91943128 0.91596635 0.90986216 0.9073793
  0.9073594  0.90513933 0.90516715 0.90301168 0.90315636 0.89761302
  0.89726084 0.8979519  0.89528124 0.8950611  0.89395933 0.89207324
  0.89233823 0.89132332 0.88814388 0.88645821 0.88558087 0.88117315
  0.8816116  0.87844306]]
[[0.         0.2203248  0.34672439 0.41756783 0.44810004 0.4693963
  0.48653644 0.49552755 0.50865508 0.52159406 0.53007764 0.53854778
  0.54640952 0.55596872 0.55918293 0.56916761 0.57284914 0.57345978
  0.58667016 0.58881126 0.59838927 0.59909822 0.6042481  0.60924671
  0.62073252 0.62088455]
 [0.         0.92865018 0.91095301 0.90942953 0.90884101 0.9080681
  0.90354421 0.90222442 0.90129221 0.89877032 0.90008889 0.90045624
  0.89982306 0.89688226 0.89413422 0.8920341  0.8957098  0.89508858
  0.89193708 0.88867502 0.88906917 0.88991576 0.88639362 0.88504375
  0.88132177 0.88037392]]
```