EC ENGR 219

2023 Winter

Project 3:

Recommender Systems

Wenxin Cheng	706070535	wenxin0319@g.ucla.edu
Yuxin Yin	606073780	yyxyy999@g.ucla.edu
Yingqian Zhao	306071513	zhaoyq99@g.ucla.edu

- **QUESTION 1: Explore the Dataset: In this question, we explore the structure of the data.**

A Compute the sparsity of the movie rating dataset:

$$\text{Sparsity} = \frac{\text{Total number of available ratings}}{\text{Total number of possible ratings}} \quad (1)$$

I. Construction Rating Matrix:

For the given datasets, we implemented them using *panda* and stored them in ratings, movies, tags and links respectively, as shown in the following code. Here each score of user i for movie j is represented by ratings.

Then we began to construct the ratings matrix, where this matrix is assumed as denoted by R. Each entry of R (i, j) represents the rating by user i for movie j and it is denoted as r_{ij}. Here we used ratings.pivot _table to construct this matrix and stored the dimensions in num_users and num_movies. The partial code is shown below,

```
1.  # constructing ratings matrix
2.  R = ratings.pivot_table(values ='rating', index ="userId", columns
    ='movieId')
3.  num_users, num_movies = R.shape
4.  print(R.shape)
5.  R.head()
```

Based on such construction, we got the following matrix table and the dimensions of such matrix is (610, 9724). Here we had 672 users and 9724 movies rated from the original datasets.

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	193581	193583	193585	193587	193609
userId																					
1	4.0	NaN	4.0	NaN	NaN	4.5	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

II. Compute the Sparsity:

After constructing the ratings matrix, we implemented Equation (1) into the following code to apply for sparsity computation.

```
1. num_possible_ratings = num_users * num_movies
2. num_available_ratings = len(ratings)
3. sparsity = num_available_ratings / num_possible_ratings
4. print("1A answer: Sparsity = {}".format(sparsity))
```

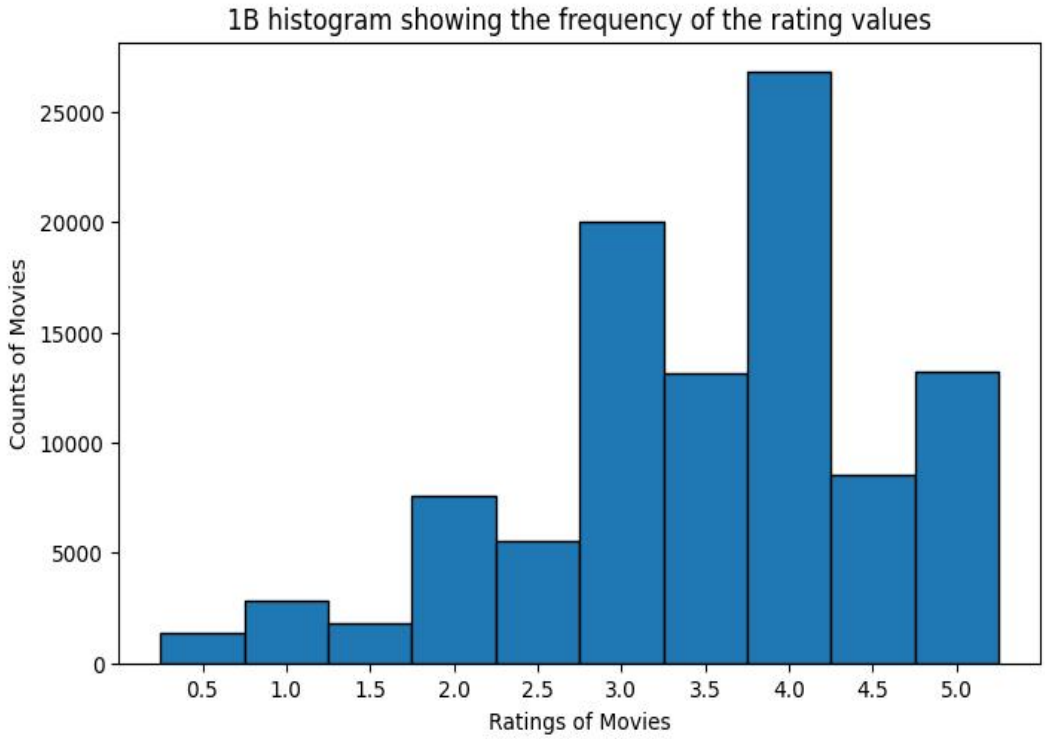
Therefore, the sparsity of the constructed ratings matrix is **0.016999683055613623**.

B Plot a histogram showing the frequency of the rating values: Bin the raw rating values into intervals of width 0.5 and use the binned rating values as the horizontal axis. Count the number of entries in the ratings matrix R that fall within each bin and

use this count as the height of the vertical axis for that particular bin. Comment on the shape of the histogram.

```
unique_ratings, unique_counts = np.unique(ratings["rating"].values,
 return_counts=True)

plt.subplots(figsize=(8,5))
plt.bar(unique_ratings, unique_counts, width=0.5, edgecolor='black',
 linewidth=1)
plt.title("1B histogram showing the frequency of the rating values"
)
plt.xlabel("Ratings of Movies")
plt.ylabel("Counts of Movies")
plt.xticks(unique_ratings)
plt.show()
```

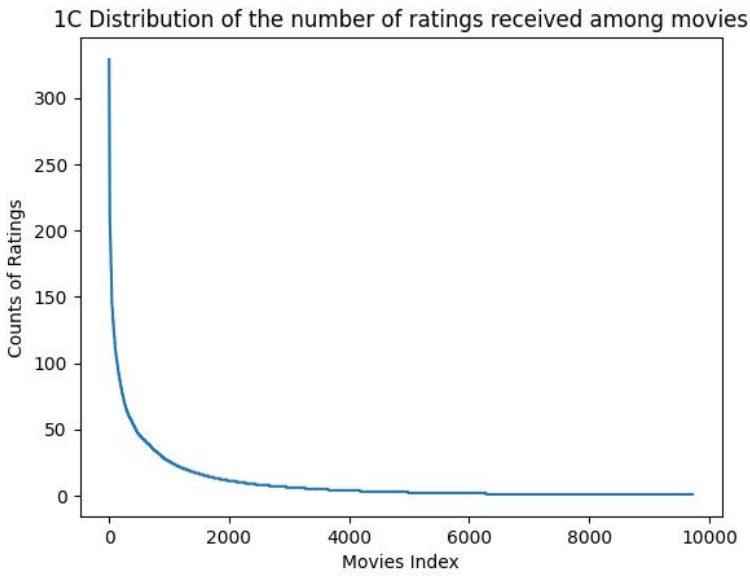
We used the above codes counting the number of entries in R and using this count as the y-lable of the histogram we obtained. Here shows the frequency of the rating values histogram as below,



As we can see in the above figure, the majority of movies are rated between 3.0 to 4.0. Besides, a very few movies are also being rated as an extremely low rate between 0.5 to 2.0. The rest of movies are being rated as a high score from 4.5 to 5.0.

C Plot the distribution of the number of ratings received among movies: The X-axis should be the movie index ordered by decreasing frequency and the Y-axis should be the number of ratings the movie has received; ties can broken in any way. A monotonically decreasing trend is expected.

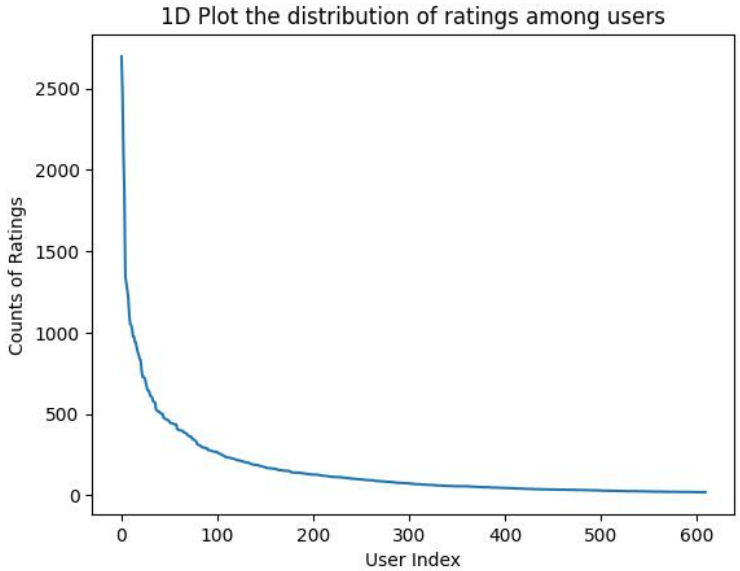
We sorted the movies by decreasing frequency index and appended each of them the number of ratings received accordingly, afterwards plotted the distribution of the number of ratings received among movies as follows,



From the figure above, we can see an apparently **monotonically decreasing trend** behaved in the distribution. As the movies' frequency decreased, the counts of ratings get smaller along with it.

D Plot the distribution of ratings among users: The X-axis should be the user index ordered by decreasing frequency and the Y-axis should be the number of movies the user has rated. The requirement of the plot is similar to that in Question C.

For plotting the trend of ratings among all users, we sorted the users by decreasing frequency index and appended the according number of movies that user has rated to it as the Y-axis. Then we obtained the following figure,

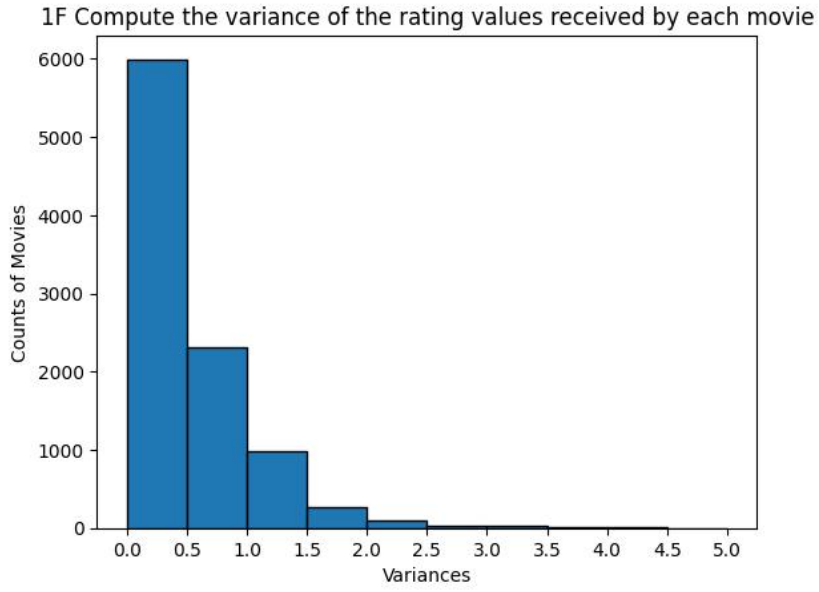


We can conclude that there is also an apparently **monotonically decreasing trend** in this plot. As the users frequency decreases, the counts of ratings also shrink.

E Discuss the salient features of the distributions from Questions C,D and their implications for the recommendation process.

From the plots in Question C and D, it can be concluded that both distributions indicate that they have a monotonically decreasing trend. It means that most of movies are not being rated but only a few movies receive the majority of ratings. Similarly, most of users do not give ratings but there are only a few users consistently rating. This accounts for the sparsity obtained in Question A, which indicates that the original constructed ratings matrix has a high sparsity and that's why we need process these data later.

F Compute the variance of the rating values received by each movie: Bin the variance values into intervals of width 0.5 and use the binned variance values as the horizontal axis. Count the number of movies with variance values in the binned intervals and use this count as the vertical axis. Briefly comment on the shape of the resulting histogram.



From the figure we obtained for this question, we can conclude that the majority of ratings for the movies are between 0.0 and 2.5. Many users share similar opinions to each movie in the datasets.

- **QUESTION 2: Understanding the Pearson Correlation Coefficient:**

A Write down the formula for μ_u in terms of I_u and r_{uk};

I_u: Set of item indices for which ratings have been specified by user u;

I_v: Set of item indices for which ratings have been specified by user v;

μ_u: Mean rating for user u computed using her specified ratings;

r_{uk}: Rating of user u for item k.

The formula should be in the way as follows,

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

B In plain words, explain the meaning of $I_u \cap I_v$. Can $I_u \cap I_v = \emptyset$? (Hint: Rating matrix R is sparse)

Here, $I_u \cap I_v$ means that the items of which user u and user v share the same opinion or have close ratings to.

Since the rating matrix R is sparse, which means that not every movie has been rated by every user, **therefore $\boldsymbol{I_u \cap I_v = \emptyset}$ will happen**. It means that user u and user v do not have any same opinions to all movies they rated, otherwise they may never rated the same movies.

- **QUESTION 3: Understanding the Prediction function:**

  Can you explain the reason behind mean-centering the raw ratings ($r_{vj} - \mu_v$) in the prediction function? (Hint: Consider users who either rate all items highly or rate all items poorly and the impact of these users on the prediction function.)

  Consider that if one user rates all items highly or rate all items poorly, and this user just has a high Pearson coefficient with the predicted user.

  This user v will have a strong difference to the expected user u. If we directly introduce user v's rating r_{vj}, this will severely inference the user u's prediction due to its extreme high or low ratings as bias in the prediction function.In this case, the bias of this user to the prediction function might be extremely high or low, which will interfere with the whole prediction results.

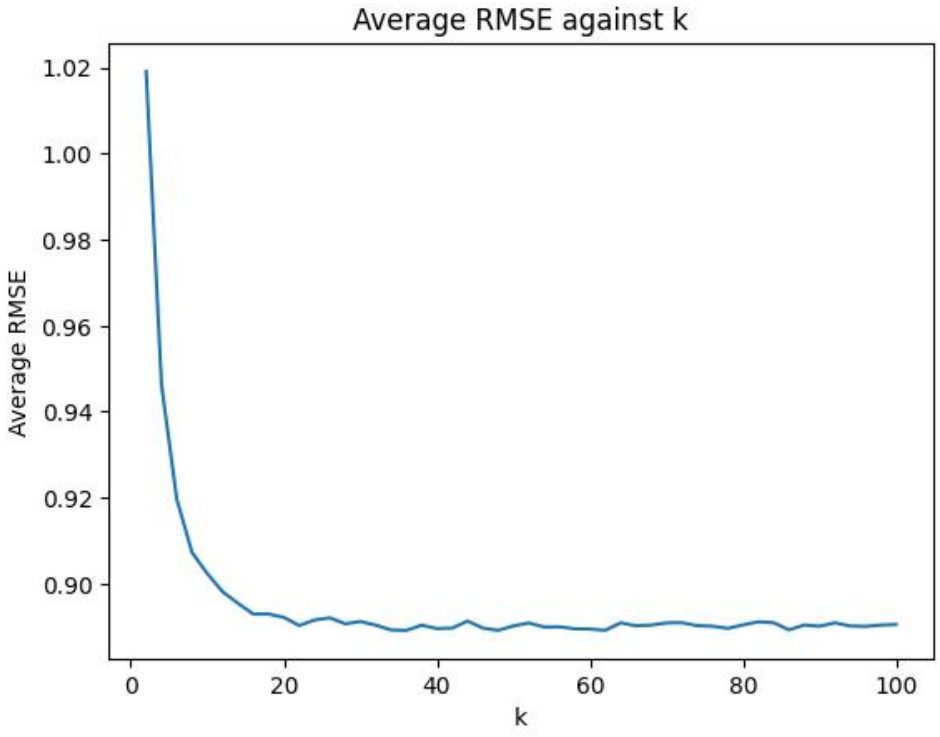
  However, the introduction to mean-centering adopts bias applied by $(r_{vj} - \mu_v)$, which can help avoid this bias implication to a huge degree.

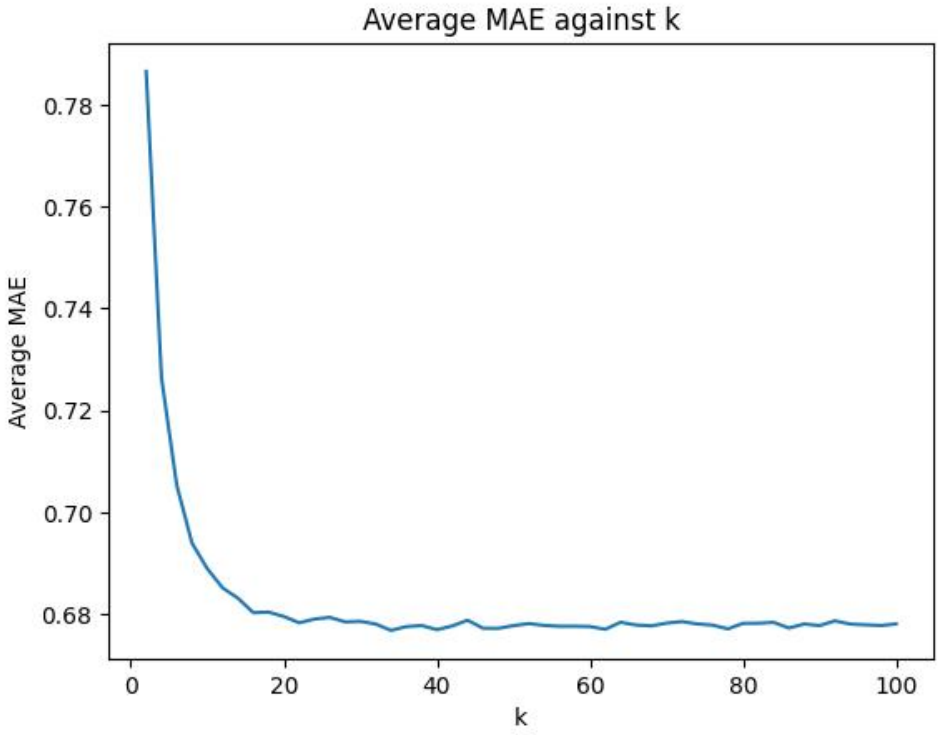
- **QUESTION 4:**

  Design a k-NN collaborative filter to predict the ratings of the movies in the original dataset and evaluate its performance using 10-fold cross validation. Sweep k (number of neighbors) from 2 to 100 in step sizes of 2, and for each k compute the average RMSE and average MAE obtained by averaging the RMSE and MAE across all 10 folds. Plot average RMSE (Y-axis) against k (X-axis) and average MAE (Y-axis) against k (X-axis).

  We designed a k-NN collaborative filter to predict the ratings of the movies in the datasets and evaluate its performance using 10-fold cross validation. To achieve the average RSME and MAE across all 10 folds, we set k sweeping from 2 to 100 in step size of 2, and for each k applied 10-fold cross validation. Using the following codes, we evaluated the k-NN collaborative filter performance and then calculated the average RMSE and average MAE over all 10 folds.

```
1.  reader = Reader(rating_scale=(0.5, 5))
2.  data = Dataset.load_from_df(ratings[['userId','movieId','rating']],
    reader=reader)
3.  avg_rmse = []
4.  avg_mae = []
5.  ks = np.arange(2,102,2)
6.
7.  for k in ks:
8.      perf = cross_validate(KNNWithMeans(k=k,sim_options={'name':'pea
    rson'}),data,cv=10)
9.      avg_rmse.append(np.mean(perf['test_rmse']))
10.     avg_mae.append(np.mean(perf['test_mae']))
```





Here we plotted average RMSE and average MAE over each k from 2 to 100 respectively, and it can be concluded that along with k increases, both average RSME and MAE decreases monotonically before k=20. When k is smaller than 20, both of the average errors have an observable amount. And after k=20, the RSME and MAE both maintain a steady state and do not change a lot no matter what change of k.

- **QUESTION 5:**

  Use the plot from question 4, to find a 'minimum k'. Note: The term 'minimum k' in this context means that increasing k above the minimum value would not result in a significant decrease in average RMSE or average MAE. If you get the plot correct, then 'minimum k' would correspond to the k value for which average RMSE and average MAE converges to a steady-state value. Please report the steady state values of average RMSE and average MAE.

```
  eps = 1e-3
  for i in range(len(ks)):
      if((abs(avg_rmse[i]-avg_rmse[i+1])<eps)):
          print(f"Minimum k for RMSE is {ks[i]}, average RMSE is {avg_rmse[i]:.3f}")
          break

  for i in range(len(ks)):
      if((abs(avg_mae[i]-avg_mae[i+1])<eps)):
          print(f"Minimum k for MAE is {ks[i]}, average MAE is {avg_mae[i]:.3f}")
          break
```

For finding the minimum k, we set value eps = 1e-3, which is used to compare with the difference between two continuous average values. The small enough value of eps let us be able to find first steady point, which is corresponded to the minimum k. And once we found this value, minimum k will be immediately printed out.

After running the above codes, we obtained that:

- RMSE

Minimum k for RMSE is 16, average RMSE is 0.893

- MAE

Minimum k for MAE is 16, average MAE is 0.680

It shows that after k =16, the RSME and MAE error decrease to a stable amount no matter how k increases. Before this minimum k, the performance of k-NN is not excellent enough since both RSME and MAE error are too high for estimating. It means that k-NN reaches a summit when k meets the minimum k and after that, the performance of k-NN will not have too much difference even increasing k to a huge amount.

- **QUESTION 6: Within EACH of the 3 trimmed subsets in the dataset, design (train and validate):**

  A k-NN collaborative filter on the ratings of the movies (i.e Popular, Unpopular or High-Variance) and evaluate each of the three models' performance using 10-fold cross validation:

  • Sweep k (number of neighbors) from 2 to 100 in step sizes of 2, and for each k compute the average RMSE obtained by averaging the RMSE across all 10 folds. Plot average RMSE (Y-axis) against k (X-axis). Also, report the minimum average RMSE.

  • Plot the ROC curves for the k-NN collaborative filters for threshold values [2.5, 3, 3.5, 4]. These thresholds are applied only on the ground truth labels in held-out validation set. For each of the plots, also report the area under the curve (AUC) value. You should have 4 × 4 plots in this section (4 trimming options – including no trimming times 4 thresholds) - all thresholds can be condensed into one plot per trimming option yielding only 4 plots.

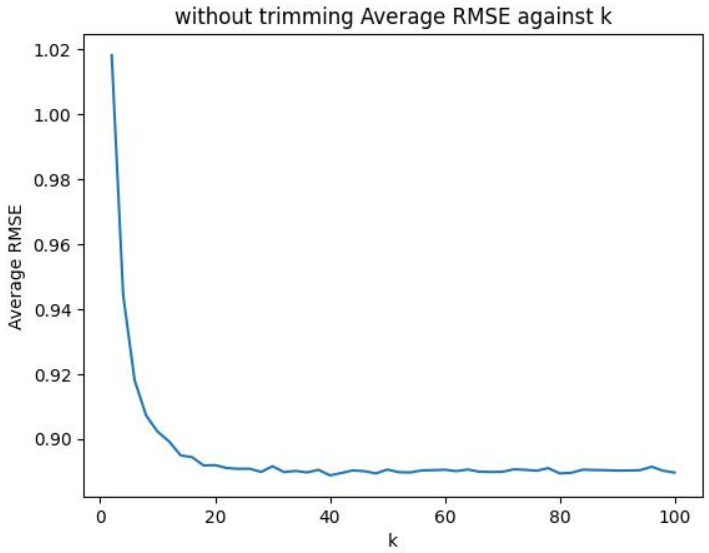
  - No Trimming

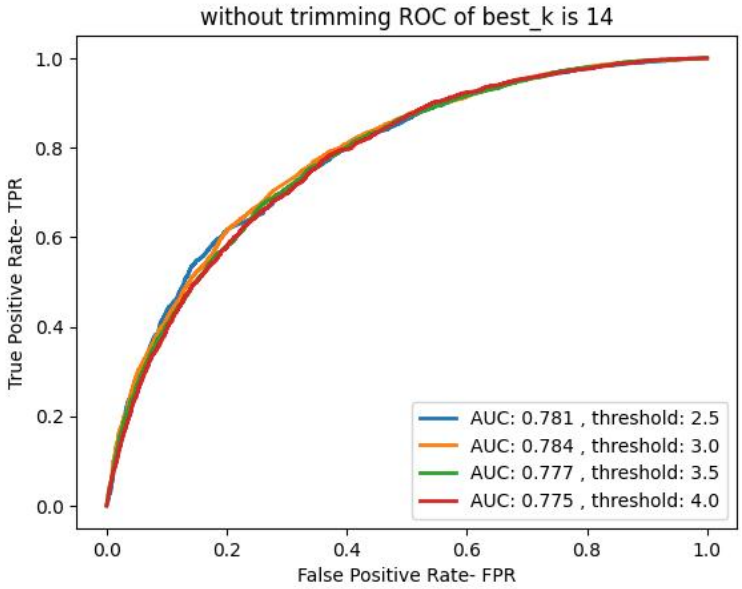
  We applied no trimming technique to the original datasets and calculated RMSE error along with k from 2 to 100. And we obtained that **without trimming Minimum k for RMSE is 14, average RMSE is 0.895.**

  After that we applied ROC curves for the k-NN collaborative filters for threshold values [2.5, 3, 3.5, 4]. And we plotted four different threshold into one figure altogether, the codes we implemented is shown below.

```
1.   def report_roc(trainset, testset, best_k, title):
2.     Threshold_list = [2.5, 3.0, 3.5, 4.0]
3.     res  = KNNWithMeans(k=best_k,sim_options={'name':'pearson'},verbo
   se=False).fit(trainset).test(testset)
4.
5.     for thre in Threshold_list:
6.       thresholded_out = []
7.       for row in res:
8.           if row.r_ui > thre:
9.               thresholded_out.append(1)
10.          else:
11.              thresholded_out.append(0)
12.      FPR, TPR, thresholds = roc_curve(thresholded_out, [row.est for
   row in res])
13.      labels = f"AUC: {auc(FPR,TPR):.3f} , threshold: {thre}"
14.      plt.plot(FPR, TPR,lw=2, label=labels)
```

After implementing this code, we plotted average RMSE error with respect to k from 2 to 100 and plotted four different threshold values' ROC curves as follows respectively:





From the ROC curve for best k = 14, we got different AUC for different threshold from [2.5, 3.0, 3.5, 4.0].

AUC: 0.781, threshold: 2.5

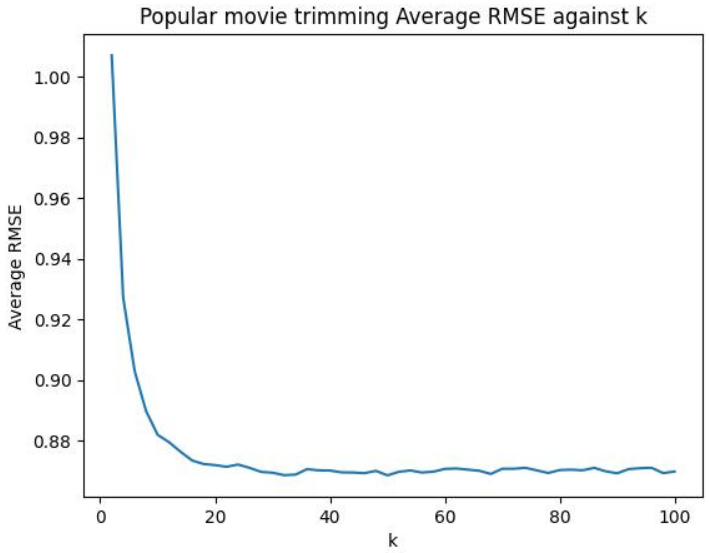
AUC: 0.784, threshold: 3.0

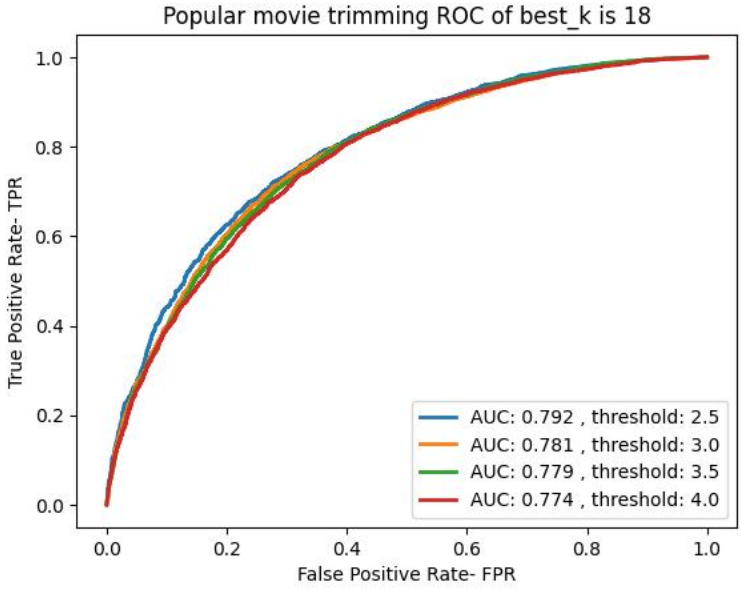
AUC: 0.777, threshold: 3.5

AUC: 0.775, threshold: 4.0

- Popular Trimming

We applied popular trimming technique to the original datasets and calculated RMSE error along with k from 2 to 100. And we obtained that **popular movie trimming Minimum k for RMSE is 18, average RMSE is 0.872.**





From the ROC curve for best k = 18, we got different AUC for different threshold from [2.5, 3.0, 3.5, 4.0].

AUC: 0.792, threshold: 2.5

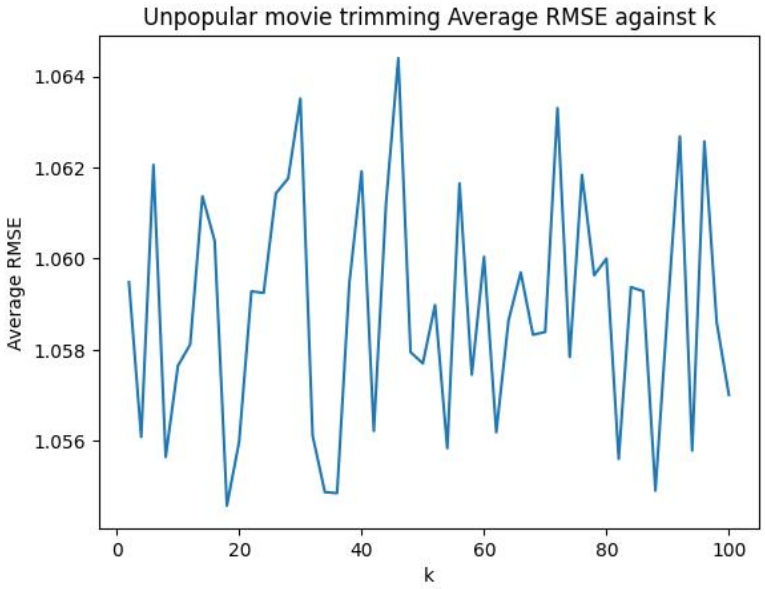
AUC: 0.781, threshold: 3.0

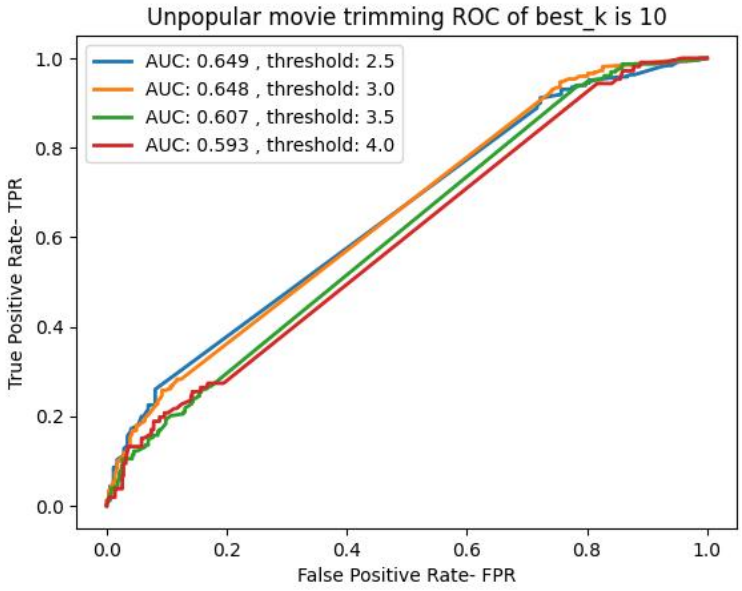
AUC: 0.779, threshold: 3.5

AUC: 0.774, threshold: 4.0

- Unpopular Trimming

We applied unpopular trimming technique to the original datasets and calculated RMSE error along with k from 2 to 100. And we obtained that **unpopular movie trimming Minimum k for RMSE is 10, average RMSE is 1.058.**





From the ROC curve for best k = 10, we got different AUC for different threshold from [2.5, 3.0, 3.5, 4.0].

AUC: 0.649, threshold: 2.5

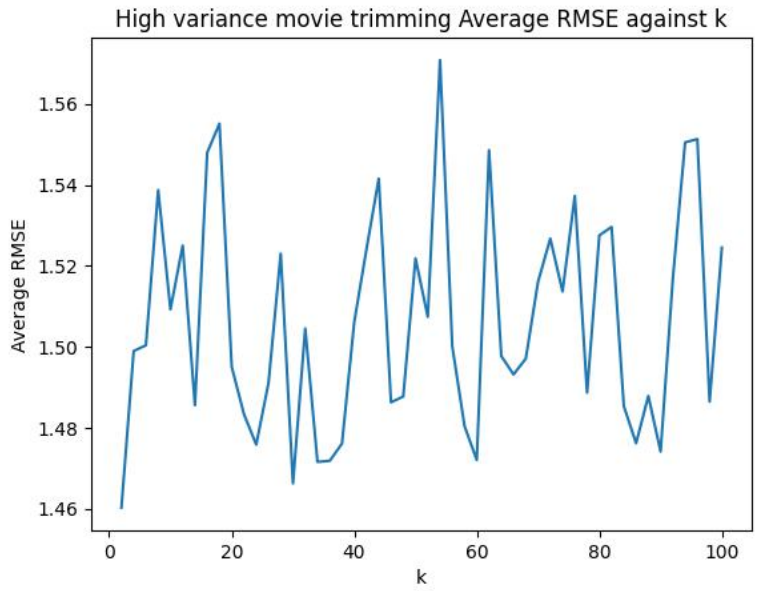
AUC: 0.648, threshold: 3.0

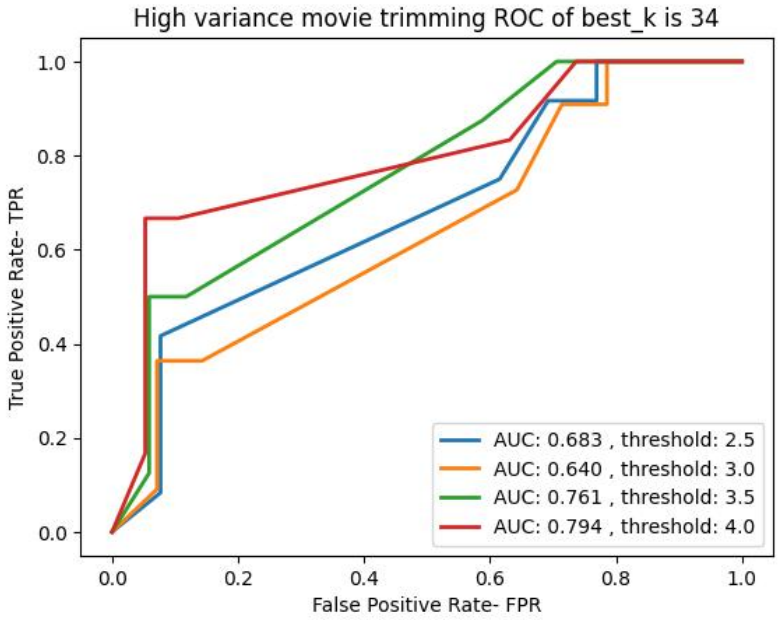
AUC: 0.607, threshold: 3.5

AUC: 0.593, threshold: 4.0

- High Variance Trimming

We applied high variance trimming technique to the original datasets and calculated RMSE error along with k from 2 to 100. And we obtained that **high variance movie trimming Minimum k for RMSE is 34, average RMSE is 1.472.**





From the ROC curve for best k = 34, we got different AUC for different threshold from [2.5, 3.0, 3.5, 4.0].

AUC: 0.683, threshold: 2.5

AUC: 0.640, threshold: 3.0

AUC: 0.761, threshold: 3.5

AUC: 0.794, threshold: 4.0

- **QUESTION 7: Understanding the NMF cost function:**

  Is the optimization problem given by equation 5 convex? Consider the optimization problem given by equation 5. For U fixed, formulate it as a least-squares problem.

  The matrix factorization problem in latent factor based model can be formulated as an optimization problem given by

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

  To determine whether this function is convex or not, let us calculate its Hessian matrix. If its Hessian matrix is not positive semi-definite, then the optimization equation would be non-convex.

  Here for convenience, we assume m=1, n=1 and $W_{11} = 1$.

  And the Hessian matrix can be written as:

$$H = \begin{bmatrix} V^2 & 2UV - R \\ 2UV - R & U^2 \end{bmatrix}$$

  Hence, the determinant of H is $-(R - UV)(R - 3UV)$, which indicates the Hessian matrix is obviously not positive semi-definite.

  From the equation above, we can see that the optimization problem **is non-convex**.

  Therefore, no matter what value m and n are set, the optimization equation will never be convex. And there will also exist many local minimums in the optimization equation.

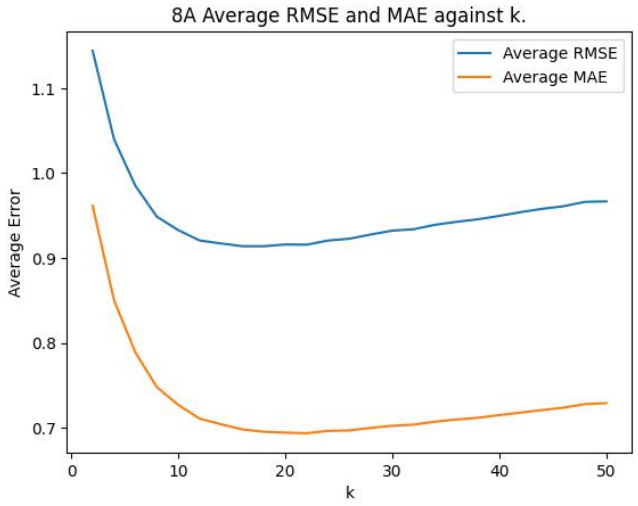
  For U fixed, the optimization problem will be transformed as follows,

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

  Here we set R is our constructed rating matrix, then we can use least-square to solve such equation. Implementing least-square is useful for converging faster and decreasing the algorithm complexity and improving the efficiency.

- **QUESTION 8: Designing the NMF Collaborative Filter:**

A Design a NMF-based collaborative filter to predict the ratings of the movies in the original dataset and evaluate its performance using 10-fold cross-validation. Sweep k (number of latent factors) from 2 to 50 in step sizes of 2, and for each k compute the average RMSE and average MAE obtained by averaging the RMSE and MAE across all 10 folds. If NMF takes too long, you can increase the step size. Increasing it too much will result in poorer granularity in your results. Plot the average RMSE (Y-axis) against k (X-axis) and the average MAE (Yaxis) against k (X-axis). For solving this question, use the default value for the regularization parameter.



B Use the plot from the previous part to find the optimal number of latent factors. Optimal number of latent factors is the value of k that gives the minimum average RMSE or the minimum average MAE. Please report the minimum average RMSE and MAE. Is the optimal number of latent factors same as the number of movie genres?

Just like what we mentioned in Question5, we calculate the minimum k if it has convergence.

```
    eps = 1e-3
    for i in range(len(ks)):
        if((abs(avg_rmse[i]-avg_rmse[i+1])<eps)):
            print(f"Minimum k for RMSE is {ks[i]}, average RMSE is {avg_rmse[i]:.3f}")
            break

    for i in range(len(ks)):
        if((abs(avg_mae[i]-avg_mae[i+1])<eps)):
            print(f"Minimum k for MAE is {ks[i]}, average MAE is {avg_mae[i]:.3f}")
            break
```

However, if we don't have a convergence K, we will choose the K with minimum RMSE.

```
    if best_k == -1: # k not convergence
```

```
        print("no convergence, we just choose a k with min rmse")
        best_k = rmse_min_k
```

RMSE: Minimum Average (NMF) = **0.914**, k = **18**
MAE: Minimum Average (NMF) = **0.694**, k = **22**
The number of latent factors tend to **be close to** the number of genres

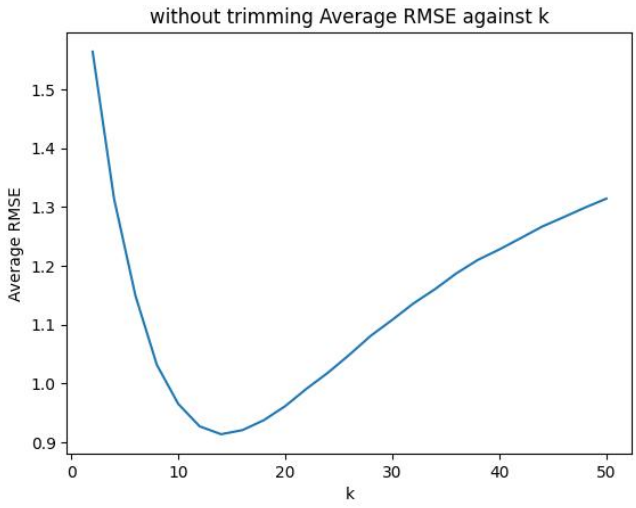
C Performance on trimmed dataset subsets: For each of Popular, Unpopular and High-Variance subsets -

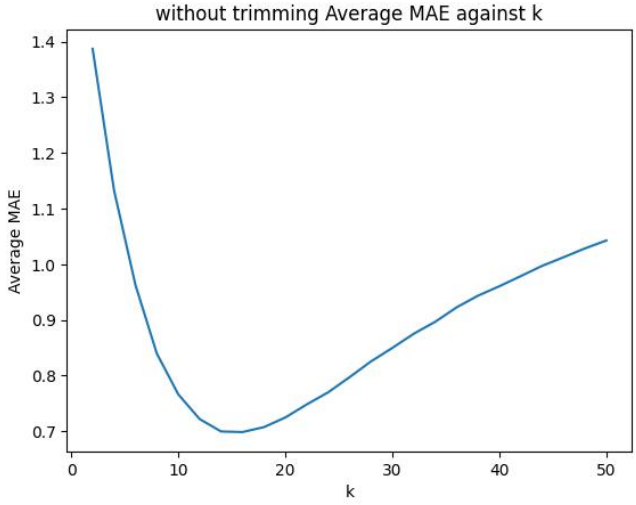
- Design a NMF collaborative filter for each trimmed subset and evaluate its performance using 10-fold cross validation. Sweep k (number of latent factors) from 2 to 50 in step sizes of 2, and for each k compute the average RMSE obtained by averaging the RMSE across all 10 folds.

- Plot average RMSE (Y-axis) against k (X-axis); item Report the minimum average RMSE.

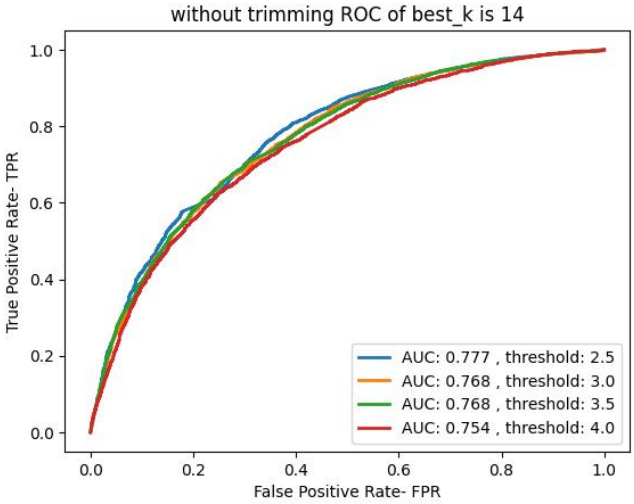
• Plot the ROC curves for the NMF-based collaborative filter and also report the area under the curve (AUC) value as done in Question 6.

Without Trimming:

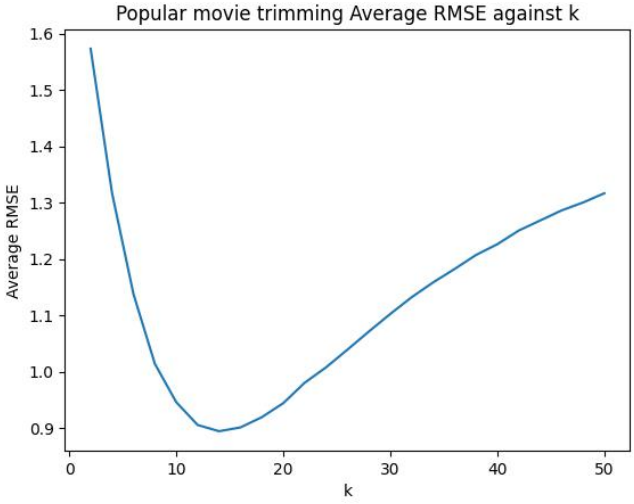


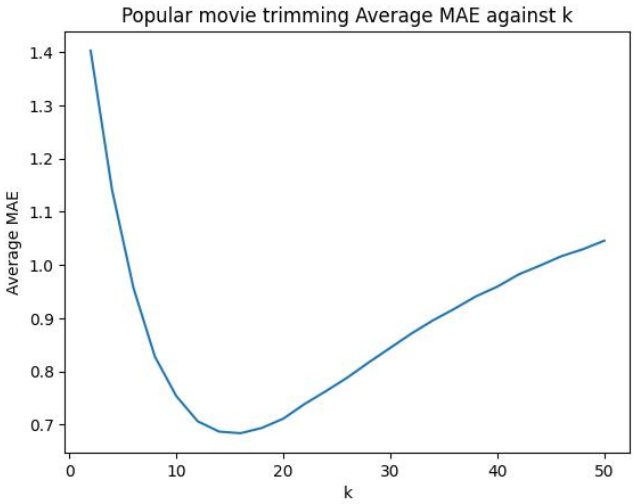


No convergence, we just choose a k with min RMSE.

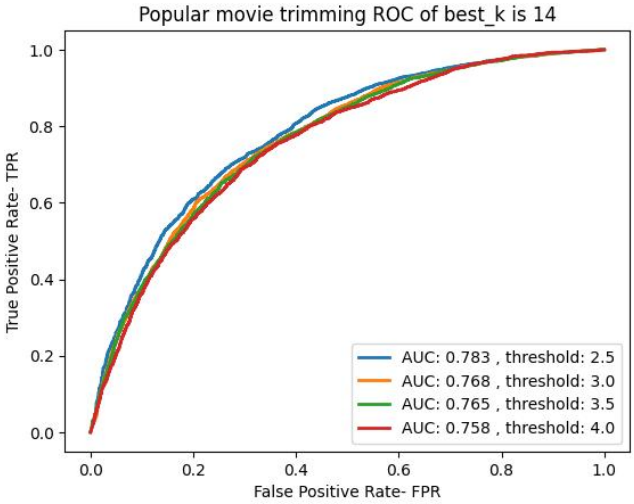


Popular Movie Trimming:

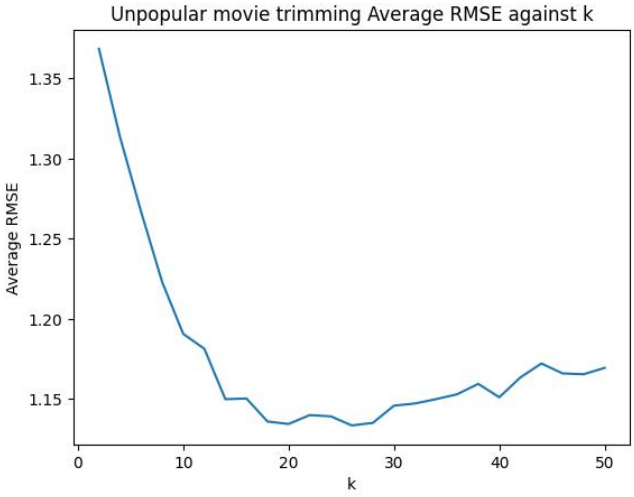


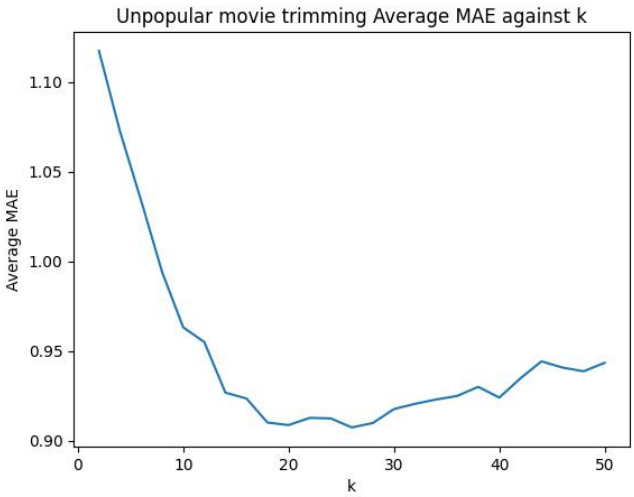


No convergence, we just choose a k with min RMSE.

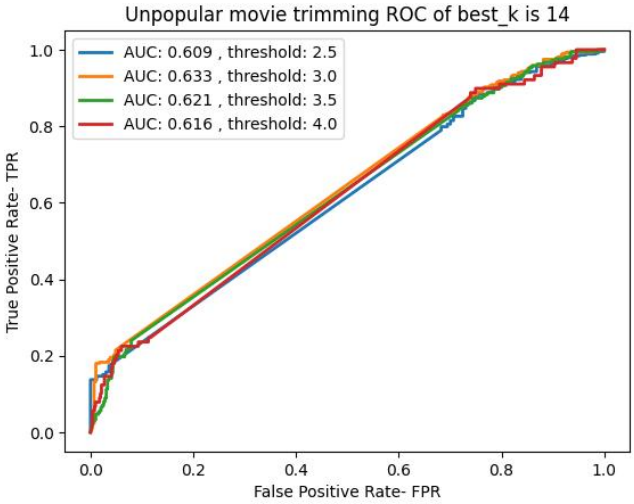


Unpopular Movie Trimming:

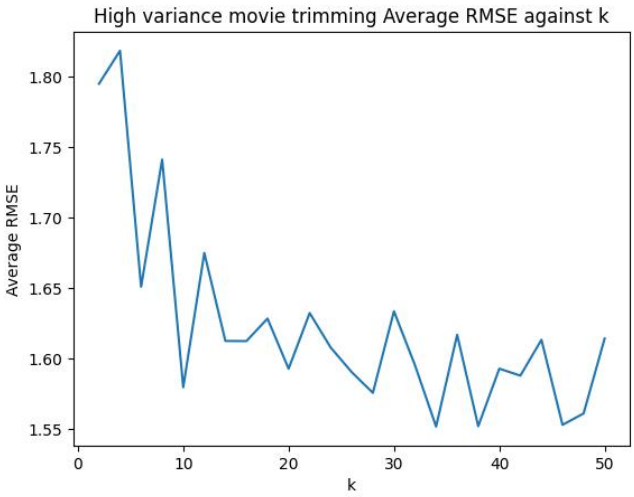


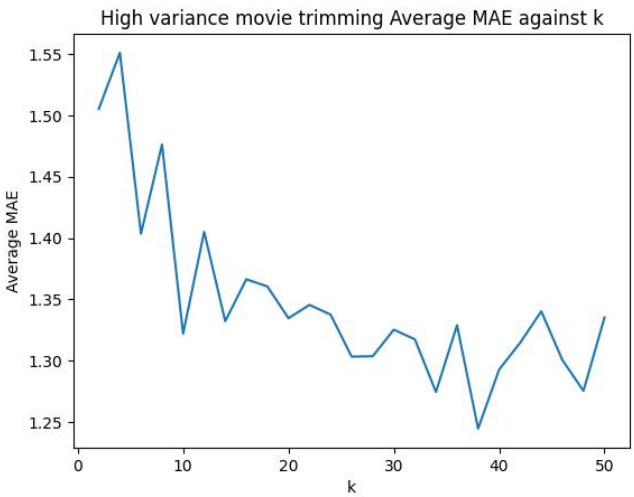


Unpopular movie trimming Minimum k for RMSE is **14**, average RMSE is **1.150**.
Unpopular movie trimming Minimum k for MAE is **22**, average MAE is **0.913**.

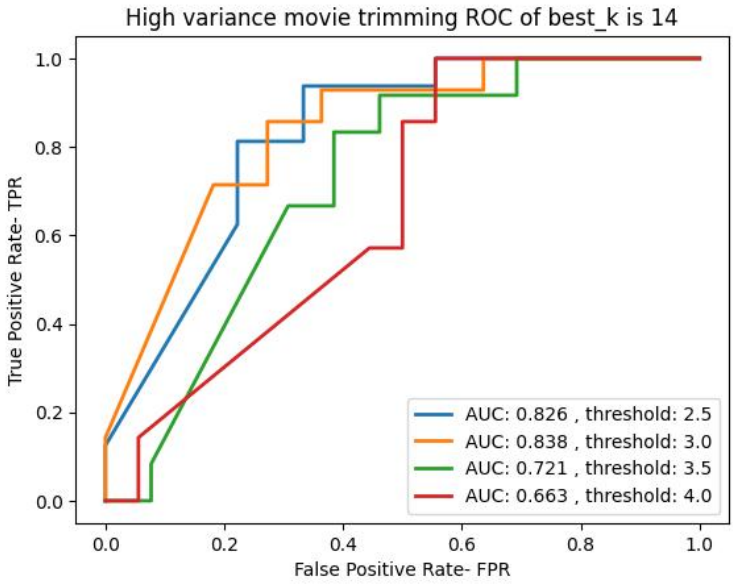


High Variance Movie Trimming:





High variance movie trimming Minimum k for RMSE is **14**, average RMSE is **1.612**.
High variance movie trimming Minimum k for MAE is **26**, average MAE is **1.303**.


High variance movie trimming ROC of best_k is 14
True Positive Rate- TPR
False Positive Rate- FPR
AUC: 0.826 , threshold: 2.5
AUC: 0.838 , threshold: 3.0
AUC: 0.721 , threshold: 3.5
AUC: 0.663 , threshold: 4.0

- **QUESTION 9: Interpreting the NMF model**

Perform Non-negative matrix factorization on the ratings matrix R to obtain the factor matrices U and V, where U represents the user-latent factors interaction and V represents the movie-latent factors interaction (use k = 20). For each column of V, sort the movies in descending order and report the genres of the top 10 movies. Do the top 10 movies belong to a particular or a small collection of genre? Is there a connection between the latent factors and the movie genres?

Latent Factor 0

Genre: Adventure|Drama Value: 2.1386
Genre: Comedy|Fantasy|Horror|Thriller Value: 1.9934
Genre: Action|Crime|Drama Value: 1.8720
Genre: Comedy|Crime|Drama Value: 1.8599
Genre: Comedy|Drama|Romance Value: 1.8268
Genre: Crime|Mystery Value: 1.7254
Genre: Horror|Thriller Value: 1.6945
Genre: Drama|Mystery|Thriller Value: 1.6634
Genre: Action|Thriller Value: 1.6603
Genre: Comedy|Romance Value: 1.6455

Latent Factor 1

Genre: Comedy Value: 2.1365
Genre: Action|Adventure|Animation|Fantasy|Sci-Fi Value: 2.0220
Genre: Comedy Value: 1.7544
Genre: Adventure|Animation|Children Value: 1.7514
Genre: Action|Animation|Children|Crime Value: 1.6294
Genre: Action|Adventure|Sci-Fi|Thriller Value: 1.6181
Genre: Comedy Value: 1.6069
Genre: Drama Value: 1.5759
Genre: Comedy Value: 1.5555
Genre: Drama Value: 1.5539

Latent Factor 2

Genre: Crime|Film-Noir Value: 2.0518
Genre: Drama|Thriller Value: 1.9304
Genre: Drama Value: 1.7626
Genre: Action|Crime|Drama Value: 1.7527
Genre: Action|Crime|Thriller Value: 1.6896
Genre: Horror|Thriller Value: 1.6653
Genre: Children|Comedy Value: 1.6411
Genre: Comedy|Romance Value: 1.6338
Genre: Drama|Film-Noir Value: 1.6238
Genre: Drama|Romance Value: 1.5968

Latent Factor 3

Genre: Drama|Romance Value: 2.4556
Genre: Comedy|Drama Value: 2.3891

Genre: Drama|Romance Value: 2.1252
Genre: Action|Comedy|IMAX Value: 1.9904
Genre: Drama Value: 1.9738
Genre: Drama Value: 1.9592
Genre: Action|Drama Value: 1.9540
Genre: Action|Crime|Drama|Mystery|Thriller Value: 1.8056
Genre: Children|Comedy|Romance Value: 1.7330
Genre: Drama|Mystery|Romance|Thriller Value: 1.6567

Latent Factor 4

Genre: Children|Drama Value: 2.5822
Genre: Comedy Value: 2.0549
Genre: Adventure|Children|Drama Value: 1.9600
Genre: Action|Adventure|Animation|Fantasy|Sci-Fi Value: 1.9284
Genre: Comedy Value: 1.9019
Genre: Children|Comedy Value: 1.8384
Genre: Drama|Romance Value: 1.8296
Genre: Action|Mystery|Sci-Fi|Thriller Value: 1.7452
Genre: Animation|Fantasy|Horror|Sci-Fi Value: 1.7419
Genre: Comedy|Drama|Romance Value: 1.7414

Latent Factor 5

Genre: Drama|Thriller Value: 2.6051
Genre: Action|Adventure|Sci-Fi Value: 2.4430
Genre: Horror|Thriller Value: 2.1767
Genre: Action|Crime|Drama Value: 1.8293
Genre: Comedy Value: 1.7877
Genre: Adventure|Comedy Value: 1.7806
Genre: Drama|Horror Value: 1.7644
Genre: Crime|Mystery Value: 1.7600
Genre: Drama Value: 1.7518
Genre: Comedy|Horror Value: 1.6550

Latent Factor 6

Genre: Comedy Value: 1.8187
Genre: Comedy|Western Value: 1.6718
Genre: Action|Adventure|Comedy|Fantasy Value: 1.5677
Genre: Horror|Sci-Fi|Thriller Value: 1.5616
Genre: Comedy|Drama Value: 1.5580
Genre: Drama Value: 1.5284
Genre: Children|Comedy|Drama Value: 1.4921
Genre: Horror|Mystery|Sci-Fi Value: 1.4859
Genre: Documentary Value: 1.4654
Genre: Comedy|Romance Value: 1.4595

Latent Factor 7

Genre: Comedy|Drama Value: 2.2399

Genre: Comedy|Drama|Romance Value: 2.0662
Genre: Crime|Drama|Fantasy|Film-Noir|Mystery|Romance Value: 1.8316
Genre: Crime|Drama|Thriller Value: 1.7977
Genre: Comedy|Fantasy|Romance Value: 1.7324
Genre: Comedy|Drama Value: 1.7093
Genre: Comedy|Drama|Romance Value: 1.7060
Genre: Comedy Value: 1.7025
Genre: Adventure|Children|Comedy|Mystery Value: 1.6586
Genre: Drama Value: 1.6545

Latent Factor 8

Genre: Comedy Value: 2.2946
Genre: Comedy Value: 2.0351
Genre: Drama|Thriller|War Value: 1.9875
Genre: Comedy|Sci-Fi Value: 1.9523
Genre: Drama Value: 1.9444
Genre: Comedy Value: 1.9270
Genre: Action|Comedy|Crime|Thriller Value: 1.8681
Genre: Comedy|Drama Value: 1.8361
Genre: Action|Thriller Value: 1.6528
Genre: Crime|Mystery|Thriller Value: 1.5259

Latent Factor 9

Genre: Documentary Value: 2.0359
Genre: Drama Value: 1.7556
Genre: Drama Value: 1.7229
Genre: Action|Crime Value: 1.6709
Genre: Comedy|Musical|Romance Value: 1.5868
Genre: Fantasy|Horror Value: 1.5861
Genre: Adventure|Fantasy|Sci-Fi Value: 1.5743
Genre: Action|Horror|Sci-Fi|Thriller Value: 1.5611
Genre: Comedy|Romance Value: 1.5609
Genre: Drama Value: 1.5580

Latent Factor 10

Genre: Comedy|Drama Value: 2.2501
Genre: Comedy Value: 1.8429
Genre: Adventure|Children|Fantasy|Sci-Fi Value: 1.7894
Genre: Drama Value: 1.7531
Genre: Adventure|Animation|Children|Drama|Fantasy Value: 1.7386
Genre: Drama|War Value: 1.7290
Genre: Action|Comedy|Drama Value: 1.7245
Genre: Drama Value: 1.6285
Genre: Drama|Sci-Fi|Thriller Value: 1.6241
Genre: Comedy|Romance Value: 1.5721

Latent Factor 11

Genre: Drama Value: 1.9061
Genre: Fantasy|Horror|Thriller Value: 1.8301
Genre: Children|Comedy Value: 1.7848
Genre: Comedy|Drama Value: 1.7680
Genre: Drama|Horror|Thriller Value: 1.7242
Genre: Drama|War Value: 1.7163
Genre: Crime|Drama|Fantasy|Film-Noir|Mystery|Romance Value: 1.6961
Genre: Drama Value: 1.6348
Genre: Comedy Value: 1.6264
Genre: Comedy Value: 1.6009

Latent Factor 12

Genre: Drama|Thriller Value: 2.5625
Genre: Drama Value: 2.3612
Genre: Adventure|Comedy|War Value: 2.2861
Genre: Adventure|Children|Drama Value: 2.0454
Genre: Comedy|Drama Value: 1.9340
Genre: Comedy Value: 1.9295
Genre: Drama|Romance Value: 1.9131
Genre: Comedy Value: 1.8981
Genre: Comedy|Crime Value: 1.8762
Genre: Drama|Romance Value: 1.8427

Latent Factor 13

Genre: Action|Mystery|Sci-Fi|Thriller Value: 2.1264
Genre: Comedy|Drama Value: 2.0597
Genre: Comedy|Romance Value: 2.0264
Genre: Comedy|Musical|Western Value: 1.9340
Genre: Drama Value: 1.9000
Genre: Drama Value: 1.8799
Genre: Children|Drama Value: 1.7453
Genre: Drama Value: 1.7422
Genre: Comedy Value: 1.7353
Genre: Drama|Horror|Thriller Value: 1.7137

Latent Factor 14

Genre: Drama Value: 2.1446
Genre: Action|Animation|Film-Noir|Sci-Fi|Thriller Value: 1.9276
Genre: Adventure|Children Value: 1.8539
Genre: Comedy|Romance Value: 1.8527
Genre: Comedy|Crime|Drama Value: 1.7733
Genre: Crime|Drama|Thriller Value: 1.7300
Genre: Adventure|Comedy|Drama|Fantasy Value: 1.6052
Genre: Comedy Value: 1.5662

Genre:
Action|Adventure|Comedy|Crime|Drama|Film-Noir|Horror|Mystery|Thriller|Western Value: **1.5316**

Genre: Comedy|Crime Value: **1.5244**

Latent Factor 15

Genre: Adventure|Drama|Sci-Fi Value: 2.7298
Genre: Action|Drama Value: 2.0109
Genre: Comedy Value: 1.9922
Genre: Documentary Value: 1.8175
Genre: Children|Comedy|Drama Value: 1.7733
Genre: Adventure|Fantasy|IMAX Value: 1.7550
Genre: Action|Crime|Drama|Thriller Value: 1.7241
Genre: Crime|Mystery Value: 1.7123
Genre: Documentary Value: 1.7098
Genre: Comedy|Romance Value: 1.7070

Latent Factor 16

Genre: Comedy|Crime Value: 1.9201
Genre: Horror|Sci-Fi|Thriller Value: 1.8048
Genre: Crime|Drama|Thriller Value: 1.7170
Genre: Documentary Value: 1.6994
Genre: Drama Value: 1.6940
Genre: Drama|Horror|Sci-Fi Value: 1.6840
Genre: Adventure|Western Value: 1.6764
Genre: Comedy|Horror Value: 1.6695
Genre: Action|Drama|Thriller Value: 1.6634
Genre: Comedy|Drama Value: 1.6441

Latent Factor 17

Genre: Comedy|Crime Value: 2.6101
Genre: Comedy Value: 1.7375
Genre: Comedy|Horror Value: 1.7210
Genre: Drama|Romance Value: 1.7036
Genre: Comedy|Romance Value: 1.6856
Genre: Action|Comedy|Sci-Fi Value: 1.6206
Genre: Action|Adventure|Drama|War Value: 1.6150
Genre: Comedy Value: 1.5919
Genre: Drama|Mystery|Sci-Fi Value: 1.5581
Genre: Action|Crime|Thriller Value: 1.5150

Latent Factor 18

Genre: Film-Noir|Mystery|Thriller Value: 2.3404
Genre: Action|Drama Value: 1.9885
Genre: Drama|Romance Value: 1.8636
Genre: Action|Adventure|Sci-Fi Value: 1.8625
Genre: Drama Value: 1.7317
Genre: Drama Value: 1.7181
Genre: Drama Value: 1.6399
Genre: Fantasy|Horror Value: 1.6232

Genre: Drama|Fantasy Value: 1.6058
Genre: Documentary|Horror Value: 1.5901

Latent Factor 19

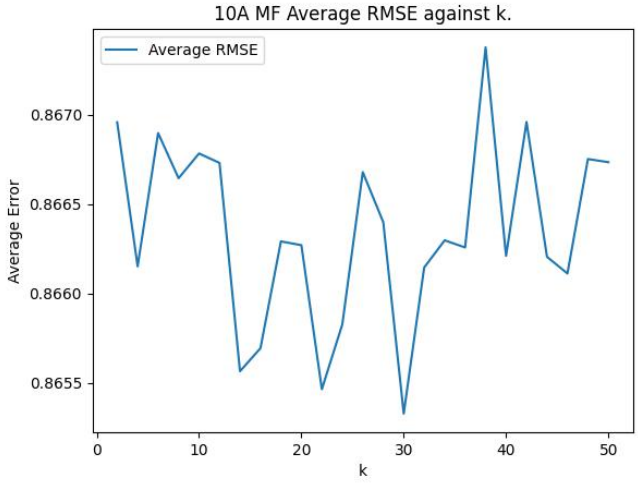
Genre: Drama|Horror Value: 2.5958
Genre: Action|Comedy Value: 2.2388
Genre: Action|Crime|Fantasy|Sci-Fi|Thriller Value: 2.1026
Genre: Horror|Sci-Fi Value: 2.0842
Genre: Drama|Romance Value: 1.9748
Genre: Crime|Drama|Mystery|Thriller Value: 1.9715
Genre: Comedy|Fantasy|Horror|Thriller Value: 1.9579
Genre: Adventure|Animation|Children|Comedy|Fantasy|Romance Value: 1.9126
Genre: Crime|Drama|Thriller Value: 1.8599
Genre: Comedy|Horror Value: 1.8531]

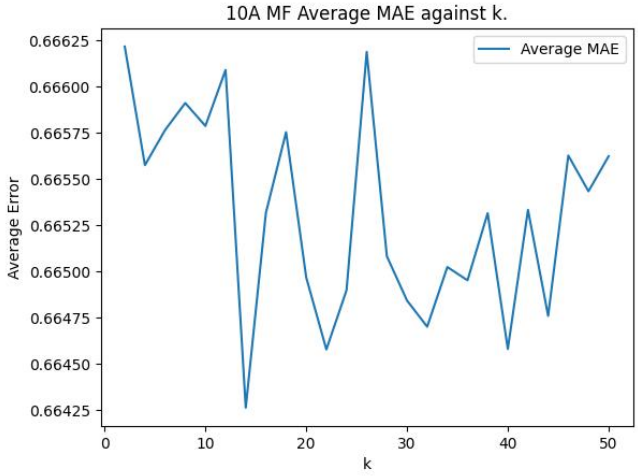
We can find that each later factor **represents different genre of movies**. NMF clusters the movies amongst the various latent factor and this clustering seems to be largely based on genre.

We can see that each later factor represents different genre of movies,When the number of latent factors increases, the number of distinct movie genres decreases. The movie genres show a better clustering when the number of latent factors increases.

- **QUESTION 10: Designing the MF Collaborative Filter**

A Design a MF-based collaborative filter to predict the ratings of the movies in the original dataset and evaluate it's performance using 10-fold cross-validation. Sweep k (number of latent factors) from 2 to 50 in step sizes of 2, and for each k compute the average RMSE and average MAE obtained by averaging the RMSE and MAE across all 10 folds. Plot the average RMSE (Y-axis) against k (X-axis) and the average MAE (Y-axis) against k (X-axis). For solving this question, use the default value for the regularization parameter.





B Use the plot from the previous part to find the optimal number of latent factors. Optimal number of latent factors is the value of k that gives the minimum average RMSE or the minimum average MAE. Please report the minimum average RMSE and MAE. Is the optimal number of latent factors same as the number of movie genres?

Minimum Average RMSE (SVD) = **0.865**, k = **24**

Minimum Average MAE (SVD) = **0.664**, k = **24**

The optimal number of latent factors is not exactly the same as the total number of movie genres. The latent components in this model are difficult to interpret.

C Performance on dataset subsets: For each of Popular, Unpopular and High-Variance subsets-

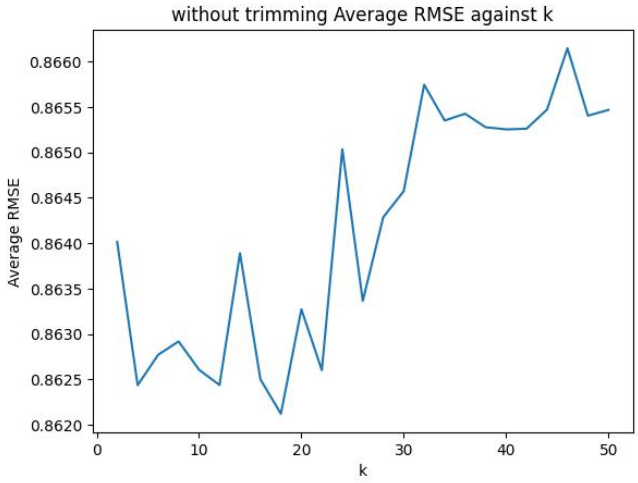
- **Design a MF collaborative filter for each trimmed subset and evaluate its**

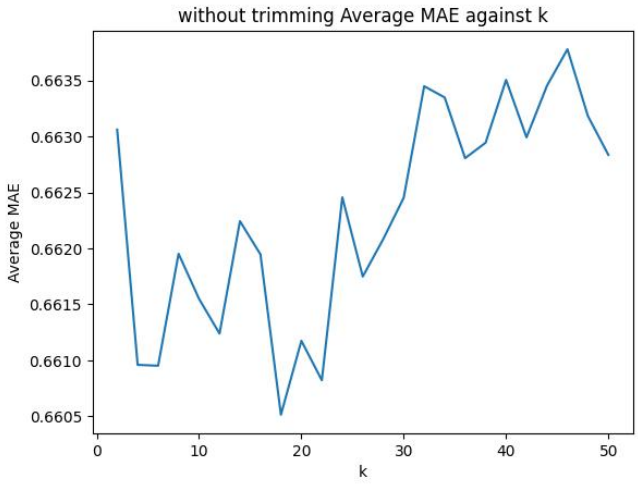
performance using 10-fold cross validation. Sweep k (number of latent factors) from 2 to 50 in step sizes of 2, and for each k compute the average RMSE obtained by averaging the RMSE across all 10 folds.

– Plot average RMSE (Y-axis) against k (X-axis); item Report the minimum average RMSE.

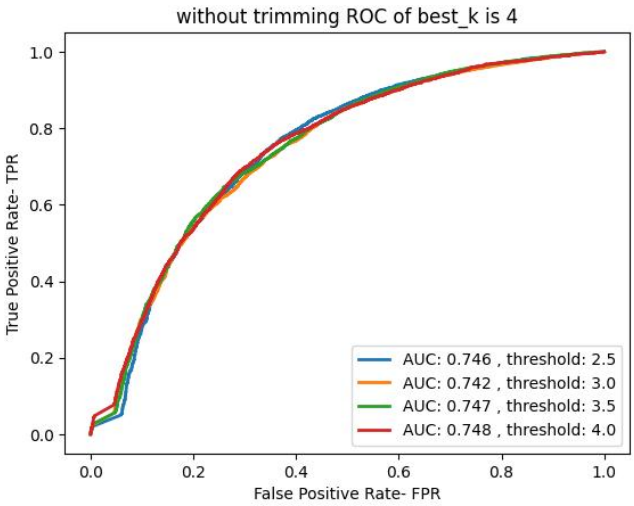
- **Plot the ROC curves for the MF-based collaborative filter and also report the area under the curve (AUC) value as done in Question 6.**

Without Trimming:

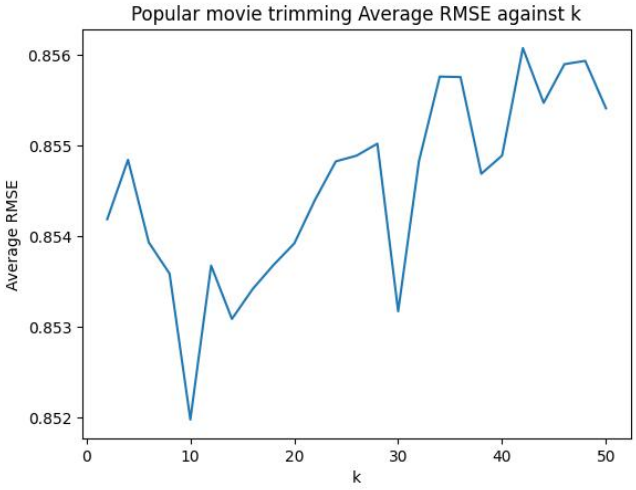


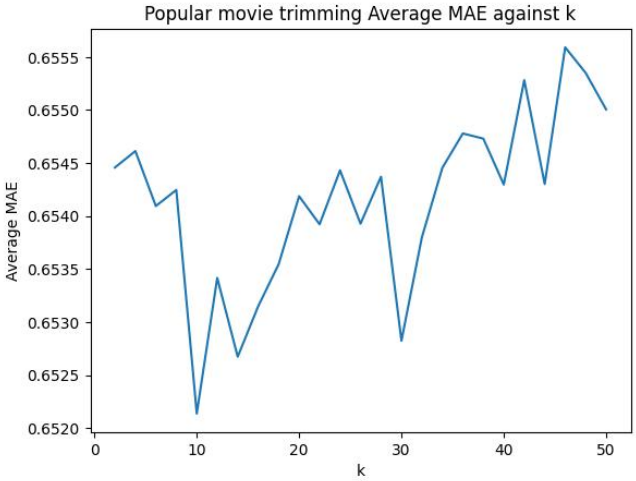


Without trimming Minimum k for RMSE is **4**, average RMSE is **0.862**.
Without trimming Minimum k for MAE is **4**, average MAE is **0.661**.

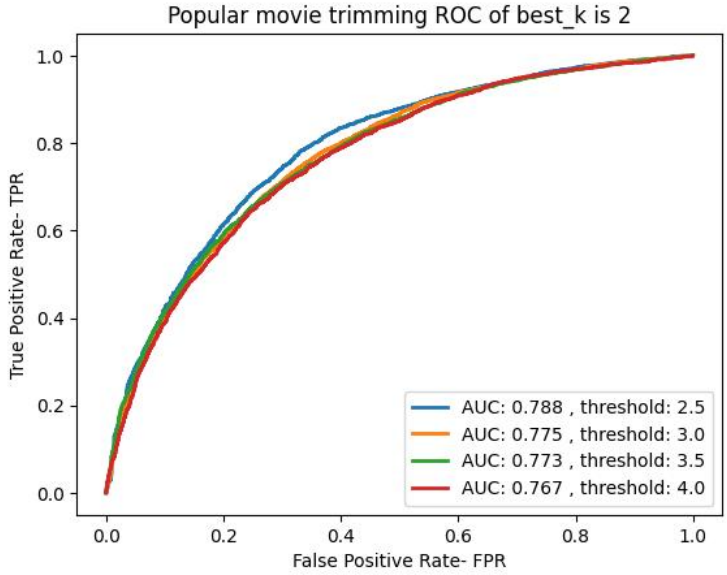


Popular Movie Trimming:

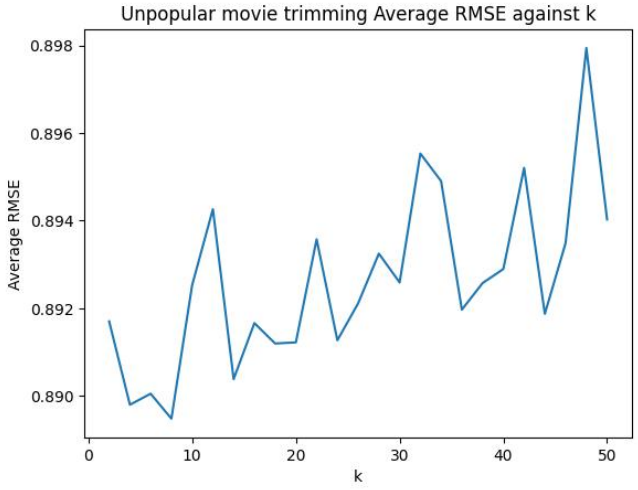


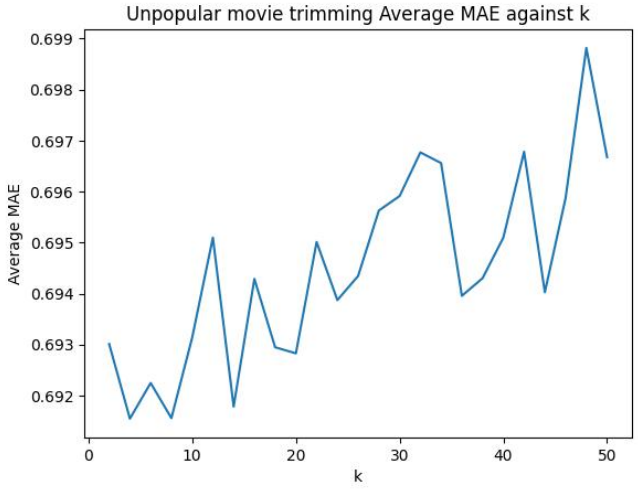


Popular movie trimming Minimum k for RMSE is **2**, average RMSE is **0.854**.
Popular movie trimming Minimum k for MAE is **2**, average MAE is **0.654**.

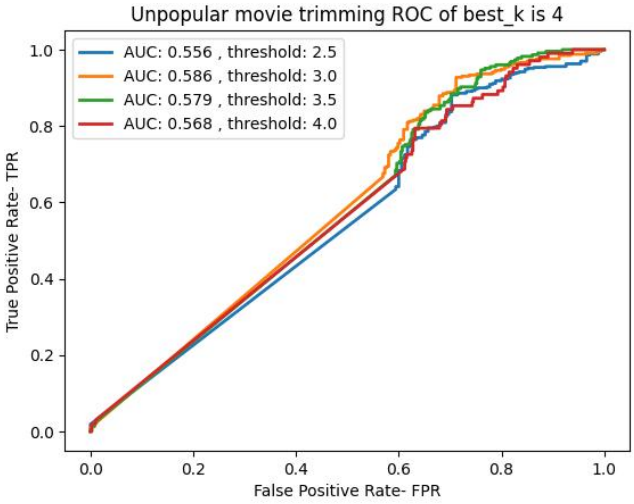


Unpopular Movie Trimming:

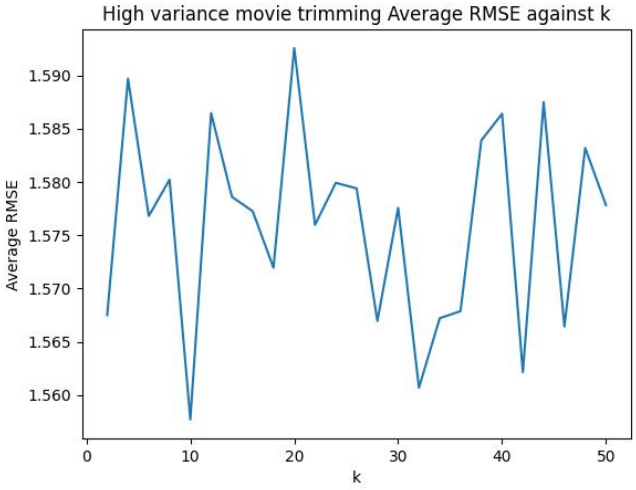


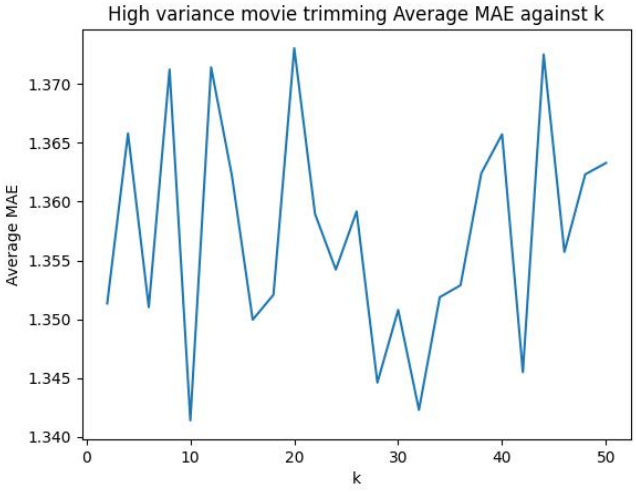


Unpopular movie trimming Minimum k for RMSE is **4**, average RMSE is **0.890**.
Unpopular movie trimming Minimum k for MAE is **4**, average MAE is **0.692**.

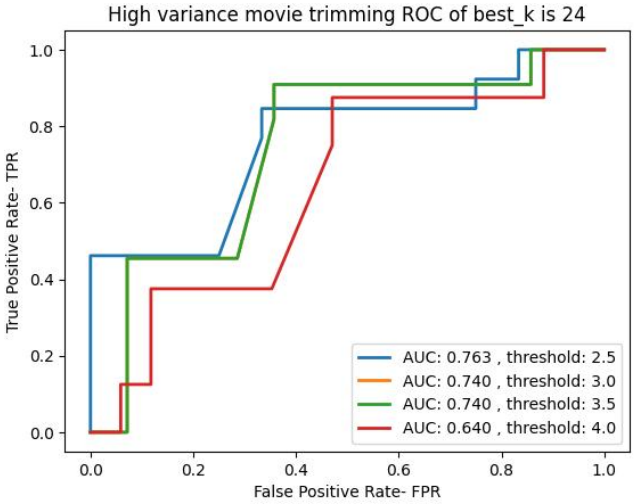


High Variance Movie Trimming:





High variance movie trimming Minimum k for RMSE is **24**, average RMSE is **1.580**.
High variance movie trimming Minimum k for MAE is **48**, average MAE is **1.362**.


High variance movie trimming ROC of best_k is 24
True Positive Rate- TPR
False Positive Rate- FPR
0.0
0.2
0.4
0.6
0.8
1.0
AUC: 0.763 , threshold: 2.5
AUC: 0.740 , threshold: 3.0
AUC: 0.740 , threshold: 3.5
AUC: 0.640 , threshold: 4.0

- **QUESTION 11: Designing a Na¨ıve Collaborative Filter:**

  • **Design a naive collaborative filter to predict the ratings of the movies in the original dataset and evaluate its performance using 10-fold cross validation. Compute the average RMSE by averaging the RMSE across all 10 folds. Report the average RMSE.**

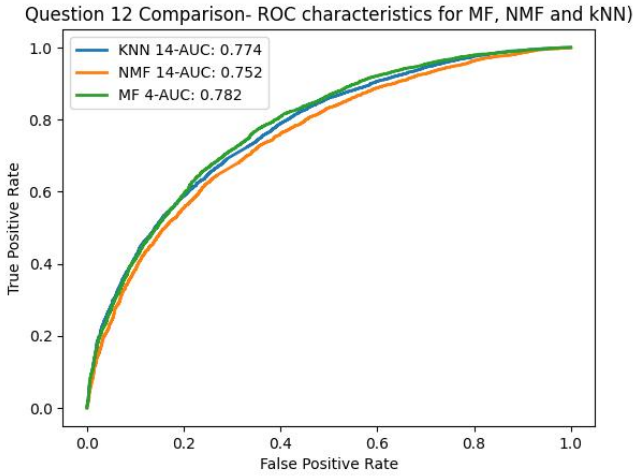
  •**Performance on dataset subsets: For each of Popular, Unpopular and High-Variance test subsets -**

  - **Design a naive collaborative filter for each trimmed set and evaluate its performance using 10-fold cross validation.**
  - **Compute the average RMSE by averaging the RMSE across all 10 folds. Report the average RMSE.**

  The average RMSE for original movie trimmed test set = **0.935**
  The average RMSE for Popular movie trimmed test set = **0.932**
  The average RMSE for Unpopular movie trimmed test set = **0.971**
  The average RMSE for High Variance movie trimmed test set = **1.455**

- **QUESTION 12: Comparing the most performant models across architecture**

  Plot the best ROC curves (threshold = 3) for the k-NN, NMF, and MF with bias based collaborative filters in the same figure. Use the figure to compare the performance of the filters in predicting the ratings of the movies.



The performance of these filters depends on the specific dataset and evaluation metric used. However, in general, MF with bias tends to outperform both K-NN and NMF in terms of prediction accuracy and robustness. MF with bias can achieve higher accuracy than K-NN due to its ability to capture latent factors and the average rating behavior of users and items. MF with bias can also outperform NMF due to its ability to handle missing values and incorporate bias terms.

- **QUESTION 13: Precision and Recall in the context of Recommender Systems**

  Precision and Recall are defined by the mathematical expressions given by equations 12 and 13 respectively. Please explain the meaning of precision and recall in your own words.

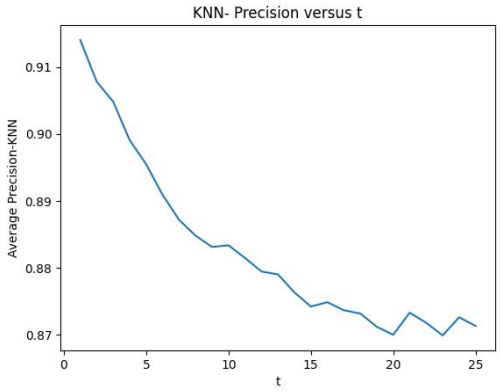
  Precision measures the proportion of recommended items that are relevant to the user's interests. It is calculated as the number of relevant items recommended to the user divided by the total number of items recommended to the user. A relevant item is one that the user has rated positively or interacted with in some way. Precision is important because it measures how accurate the recommendations are in terms of the user's interests.

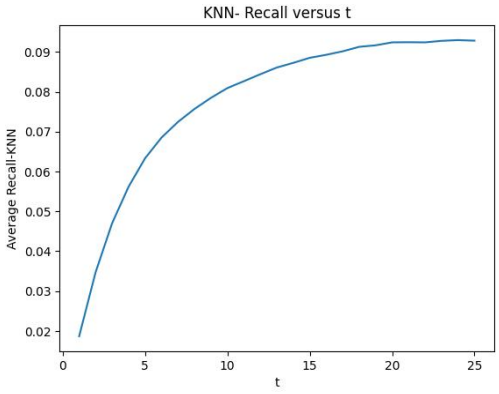
  Recall measures the proportion of relevant items that were recommended to the user. It is calculated as the number of relevant items recommended to the user divided by the total number of relevant items in the dataset. Recall is important because it measures how comprehensive the recommendations are in terms of the user's interests. Precision and recall are two common evaluation metrics used in recommender systems to measure the quality of recommendations.

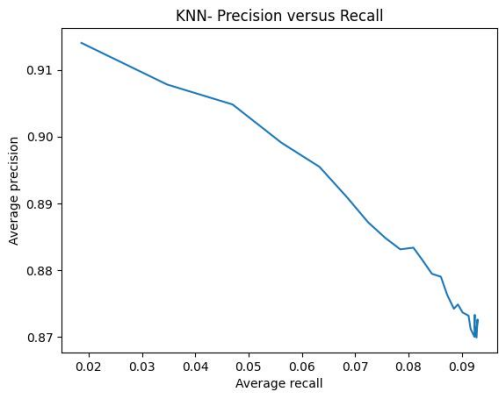
  In a word, precision is a measure of **how well the recommender system avoids recommending irrelevant items**, while recall is a measure of **how well the recommender system recommends all relevant items**. Both precision and recall are important in evaluating the effectiveness of a recommender system. However, they are often trade-offs between them.

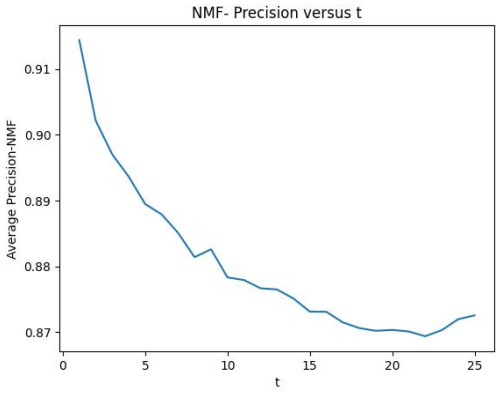
- **QUESTION 14: Comparing the precision-recall metrics for the different models**
  - **For each of the three architectures:**
    - **Plot average precision (Y-axis) against t (X-axis) for the ranking obtained using the model's predictions.**
    - **Plot the average recall (Y-axis) against t (X-axis) and plot the average precision (Y-axis) against average recall (X-axis).**
    - **Use the best k found in the previous parts and sweep t from 1 to 25 in step sizes of 1. For each plot, briefly comment on the shape of the plot.**

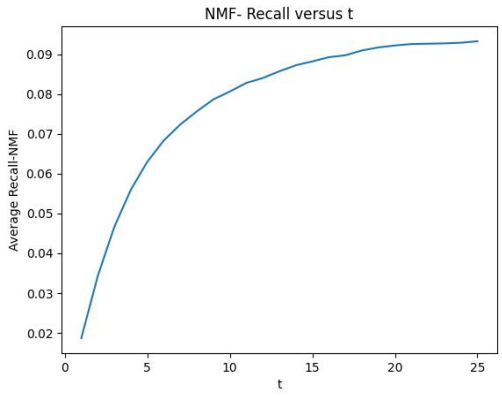
- **Plot the best precision-recall curves obtained for the three models (k-NN, NMF, MF) in the same figure. Use this figure to compare the relevance of the recommendation list generated using k-NN, NMF, and MF with bias predictions.**

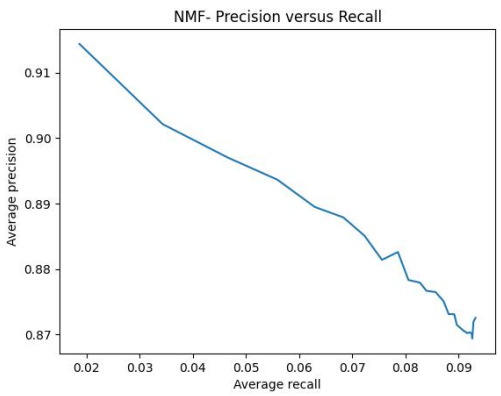


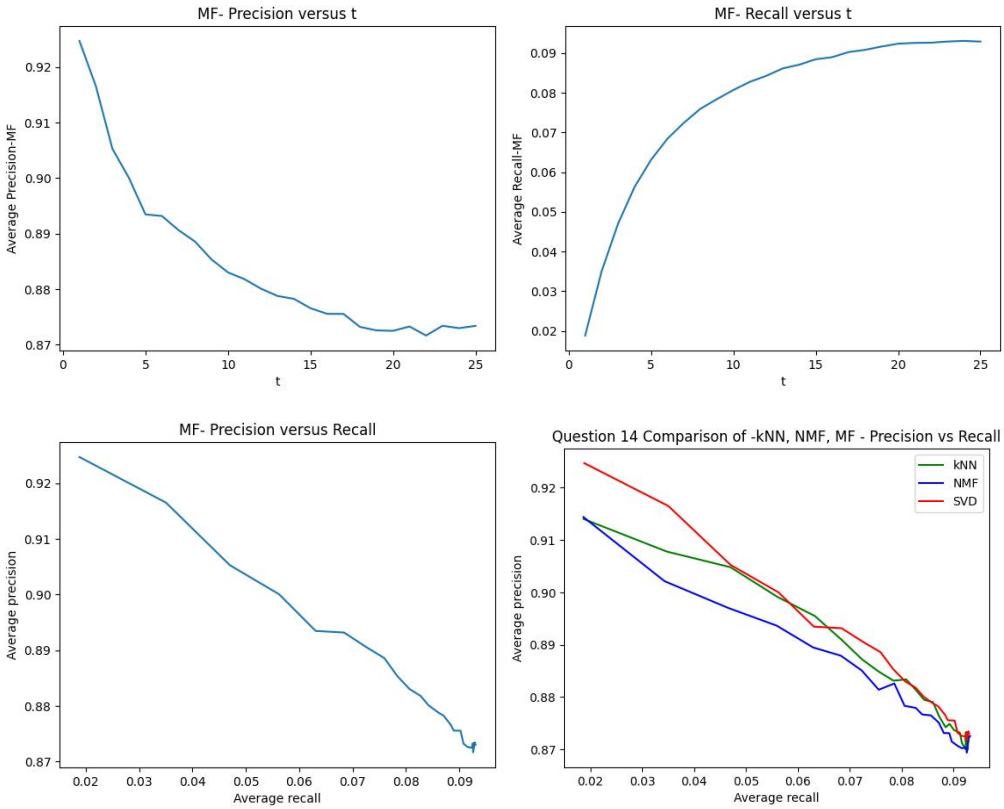












The relevance of the recommendation list generated by k-NN, NMF, and MF with bias depends on the specific dataset and evaluation metric used.

K-NN is a simple and intuitive algorithm for generating recommendations, but it suffers from the cold start problem. Furthermore, k-NN is sensitive to the choice of distance metric and neighborhood size, which can impact the quality of the recommendations.

NMF can handle missing values and can provide interpretable latent factors. However, it requires a large amount of computation and can be sensitive to the initialization of the latent factors. NMF can also suffer from overfitting if the dataset is small or noisy.

MF with bias can handle missing values and is less sensitive to the initialization of the latent factors than NMF. MF with bias can also incorporate implicit feedback, such as clicks or views, into the rating predictions, which can further improve the relevance of the recommendations.

In general, MF with bias tends to **generate more relevant recommendations** than k-NN and NMF because it can capture both explicit and implicit feedback, handle missing values and incorporate bias terms.

- **Conclusion**

```
1.    def popular_trimming(row):
2.      return len(ref[row['movieId']]) > 2
3.    def unpopular_trimming(row):
4.      return len(ref[row['movieId']]) <= 2
5.    def high_variance_trimming(row):
6.      return len(ref[row['movieId']]) >= 5 and row['movieId'] in varian
   ce[variance>=2].keys()
7.    def do_nothing(row):
8.      return True
9.
10.   filter_ratings = ratings[ratings.apply(filter, axis=1)]
11.   data = Dataset.load_from_df(filter_ratings[['userId','movieId','rat
   ing']], reader=reader)
```

In this project and code implementation, we find that we can not simply apply the function to filter the trainset if we do the k-fold split first, so we change to “apply” function by returning true/false to filter the dataset.

In this project, we implement and analyze the performance of two types of collaborative filtering methods of recommendation systems

1. Neighborhood-based collaborative filtering: Directly leverages the choices of other

users to determine potential items to recommend to the current user.

2. Model-based collaborative filtering: Estimates a joint model from all the user data

such that in order to generate a new recommendation, we do not need to use the entire

user base and can query (a smaller) model.

- **Work Distribution**

Our group members are evenly distributed.

Project 3: Recommender Systems

Wenxin Cheng 706070535 wenxin0319@g.ucla.edu

Yuxin Yin 606073780 yyxyy999@g.ucla.edu

Yingqian Zhao 306071513 zhaoyq99@g.ucla.edu

0 pre-install packages

In [1]:
```
# !conda install -c conda-forge scikit-surprise
```

Question 1

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from collections import defaultdict

from surprise import Reader, Dataset, accuracy
from surprise.prediction_algorithms.knns import KNNWithMeans
from surprise.model_selection import cross_validate, KFold, train_test_split
from sklearn.metrics import roc_curve, auc, mean_squared_error
from surprise.prediction_algorithms.matrix_factorization import NMF, SVD
from tqdm import tqdm, trange
```

In [3]:
```
ratings = pd.read_csv("Synthetic_Movie_Lens/ratings.csv")
movies = pd.read_csv("Synthetic_Movie_Lens/movies.csv")
tags = pd.read_csv("Synthetic_Movie_Lens/tags.csv")
links = pd.read_csv("Synthetic_Movie_Lens/links.csv")
```

In [4]:
```
# constructing ratings matrix
R = ratings.pivot_table(values ='rating', index ="userId", columns ='movieId')
num_users, num_movies = R.shape
print(R.shape)
R.head()
```

```
(610, 9724)
```

Out[4]:

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573
userId															
1	4.0	NaN	4.0	NaN	NaN	4.5	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
5	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN

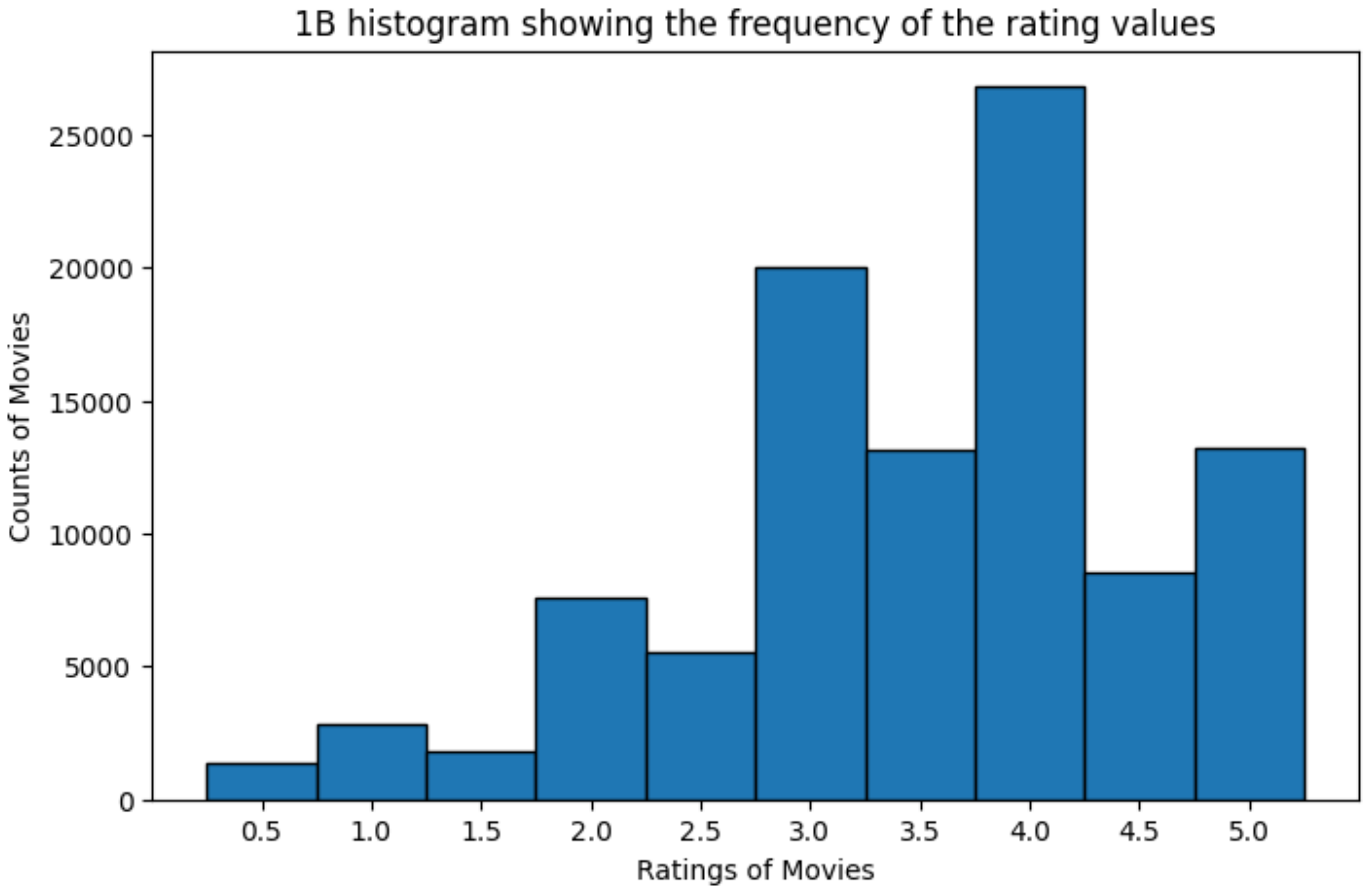
5 rows × 9724 columns

In [5]:
```
num_possible_ratings = num_users * num_movies
num_available_ratings = len(ratings)
```

```
sparsity = num_available_ratings / num_possible_ratings
print("1A answer: Sparsity = {}".format(sparsity))
```

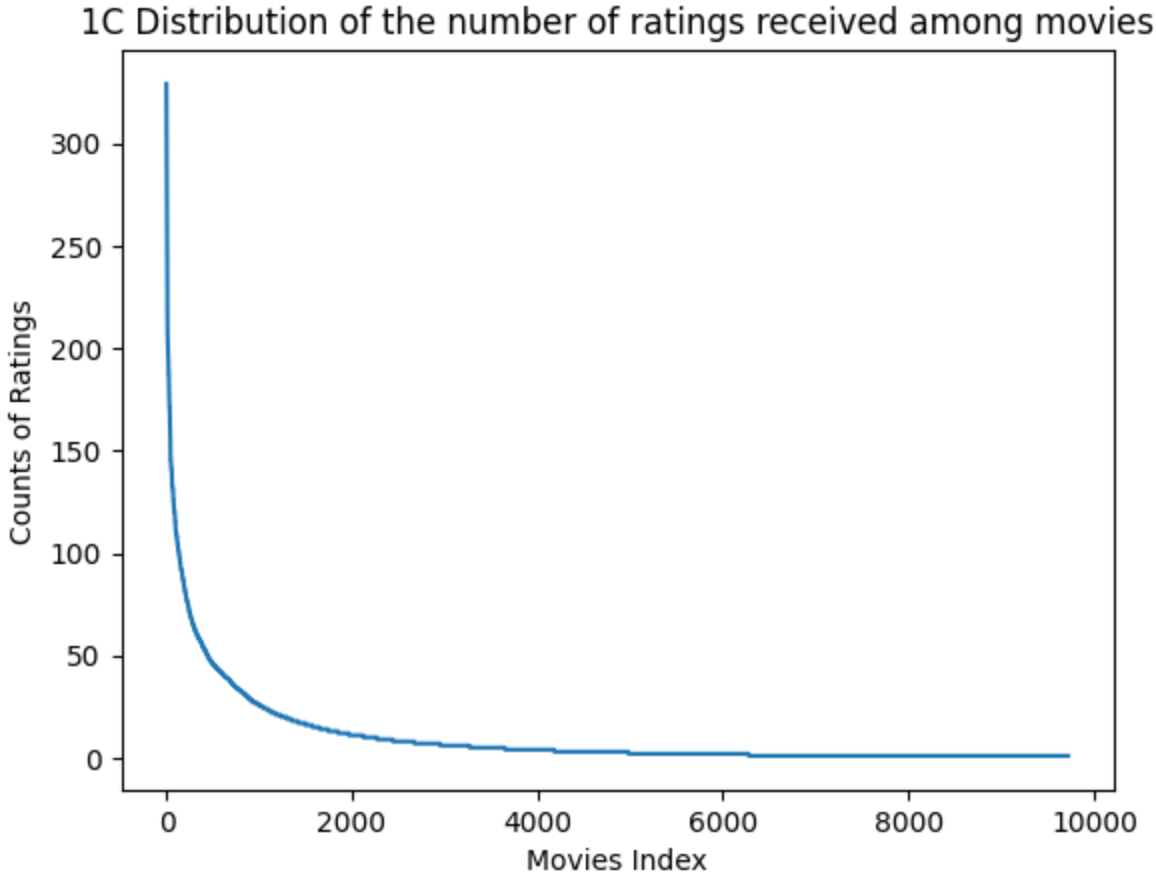
```
1A answer: Sparsity = 0.016999683055613623
```

In [6]:
```
unique_ratings, unique_counts = np.unique(ratings["rating"].values, return_counts=True)

plt.subplots(figsize=(8,5))
plt.bar(unique_ratings, unique_counts, width=0.5, edgecolor='black', linewidth=1)
plt.title("1B histogram showing the frequency of the rating values")
plt.xlabel("Ratings of Movies")
plt.ylabel("Counts of Movies")
plt.xticks(unique_ratings)
plt.show()
```



In [7]:
```
movie_value = (num_users- np.isnan(R).sum()).sort_values(ascending=False)
print(movie_value)
plt.plot(movie_value.values)
plt.title("1C Distribution of the number of ratings received among movies")
plt.xlabel("Movies Index")
plt.ylabel("Counts of Ratings")
plt.show()
```

```
movieId
356       329
318       317
296       307
593       279
2571      278
         ...
4093        1
4089        1
58351       1
4083        1
193609      1
Length: 9724, dtype: int64
```

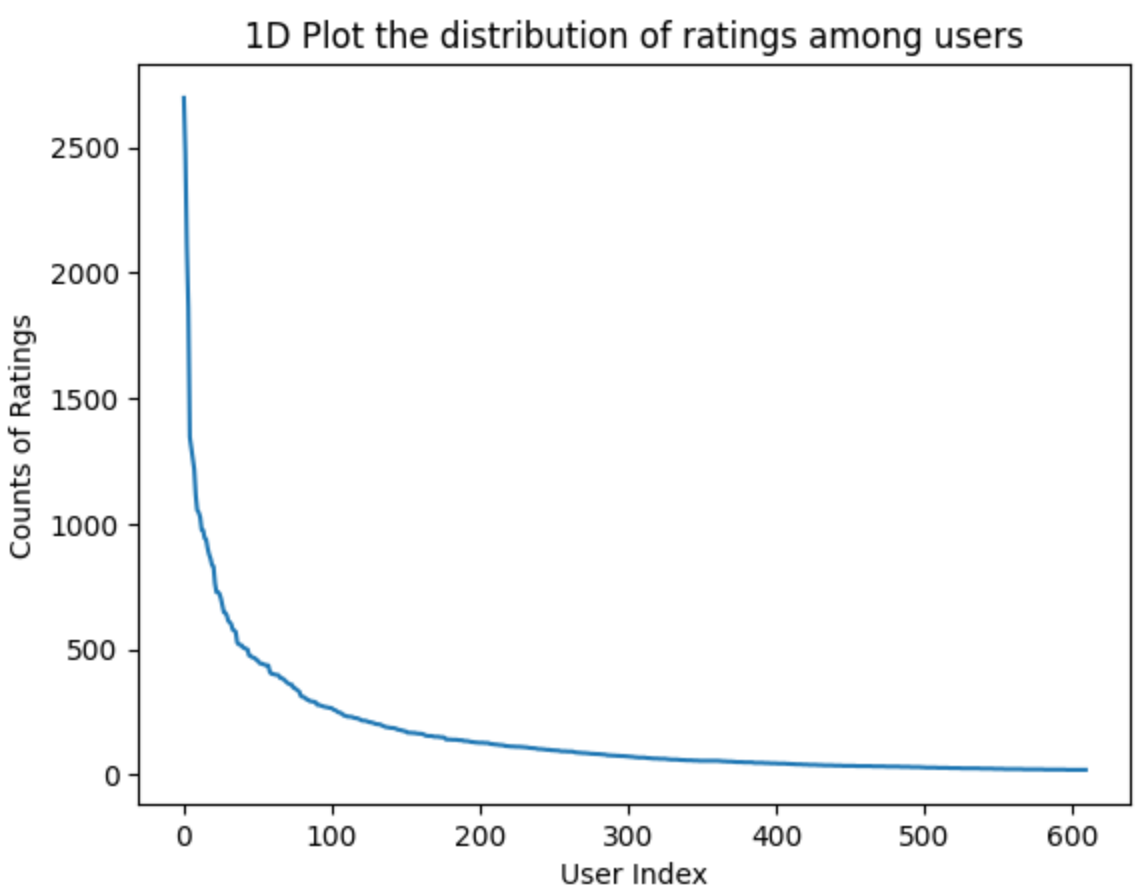


In [8]:
```
print(R.T.shape)
```

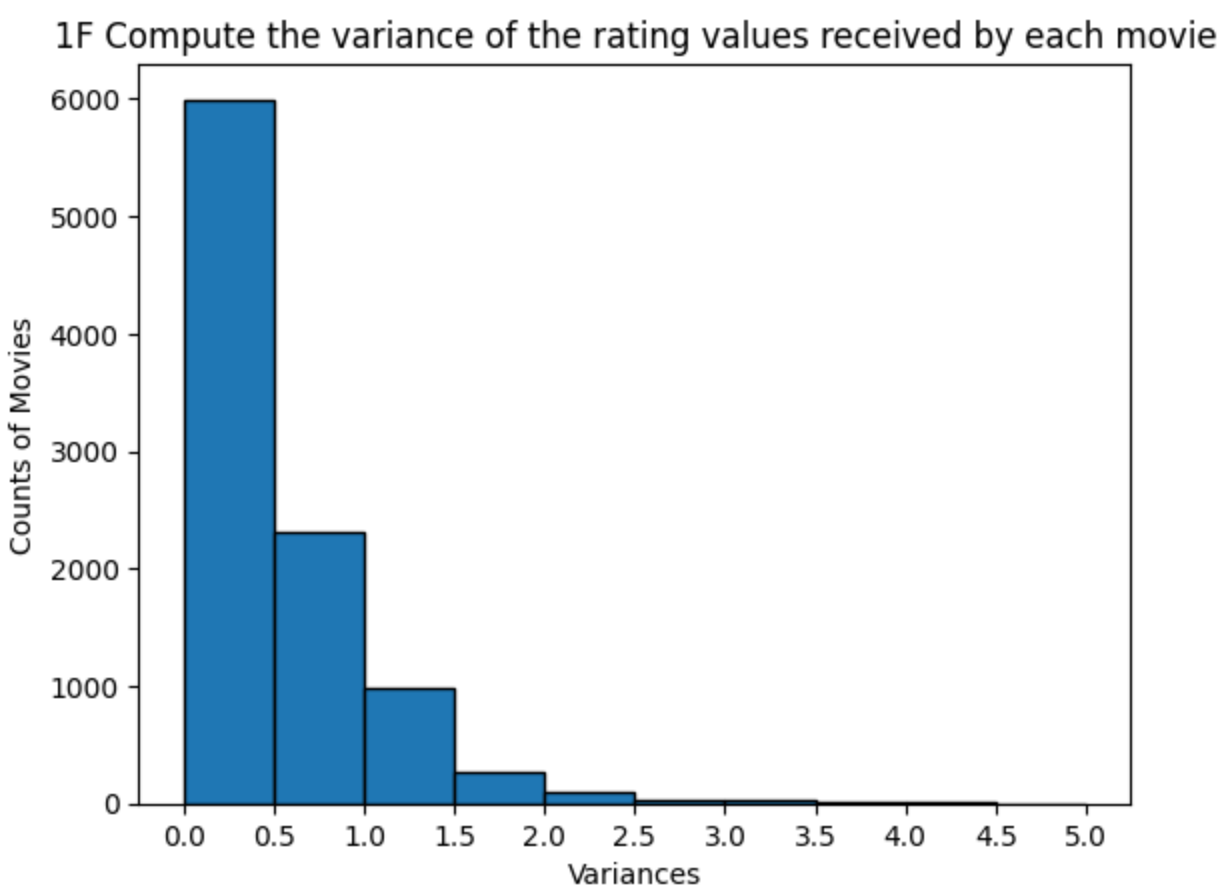
```
(9724, 610)
```

In [9]:
```
user_value = (num_movies- np.isnan(R.T).sum()).sort_values(ascending=False)
print(user_value)
plt.plot(user_value.values)
plt.title("1D Plot the distribution of ratings among users")
plt.xlabel("User Index")
plt.ylabel("Counts of Ratings")
plt.show()
```

```
userId
414    2698
599    2478
474    2108
448    1864
274    1346
       ...
442      20
569      20
320      20
576      20
53       20
Length: 610, dtype: int64
```



In [10]:
```
rates = np.linspace(0,5,num=11)
plt.hist(np.var(ratings.pivot_table("rating", "userId", "movieId")).values,bins=rates,ed
plt.title("1F Compute the variance of the rating values received by each movie")
plt.xlabel("Variances")
plt.ylabel("Counts of Movies")
plt.xticks(rates)
plt.show()
```



Question 2 and 3 are stated on our report

Question 4

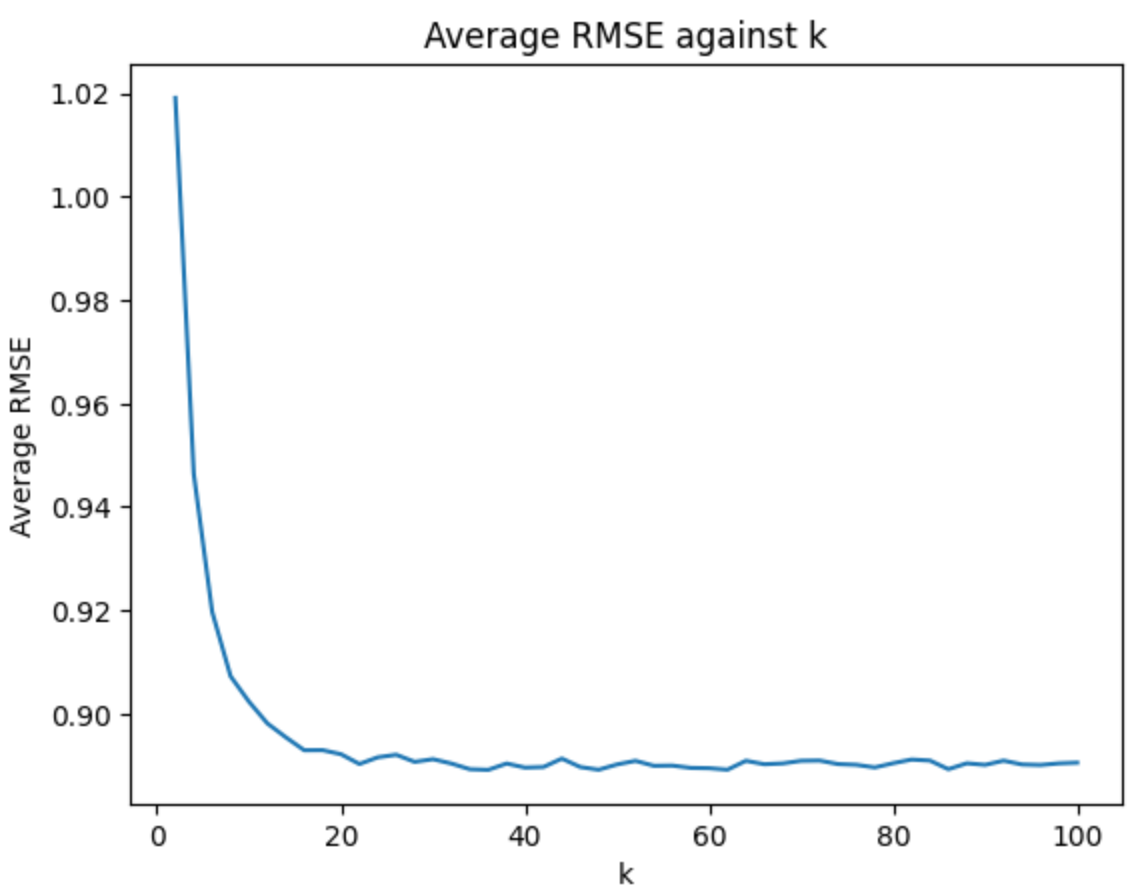
In [11]:
```
reader = Reader(rating_scale=(0.5, 5))
data = Dataset.load_from_df(ratings[['userId','movieId','rating']], reader=reader)

avg_rmse = []
avg_mae = []
ks = np.arange(2,102,2)

for k in ks:
    perf = cross_validate(KNNWithMeans(k=k,sim_options={'name':'pearson'}),data,cv=10)
    avg_rmse.append(np.mean(perf['test_rmse']))
    avg_mae.append(np.mean(perf['test_mae']))
```

```
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.
```

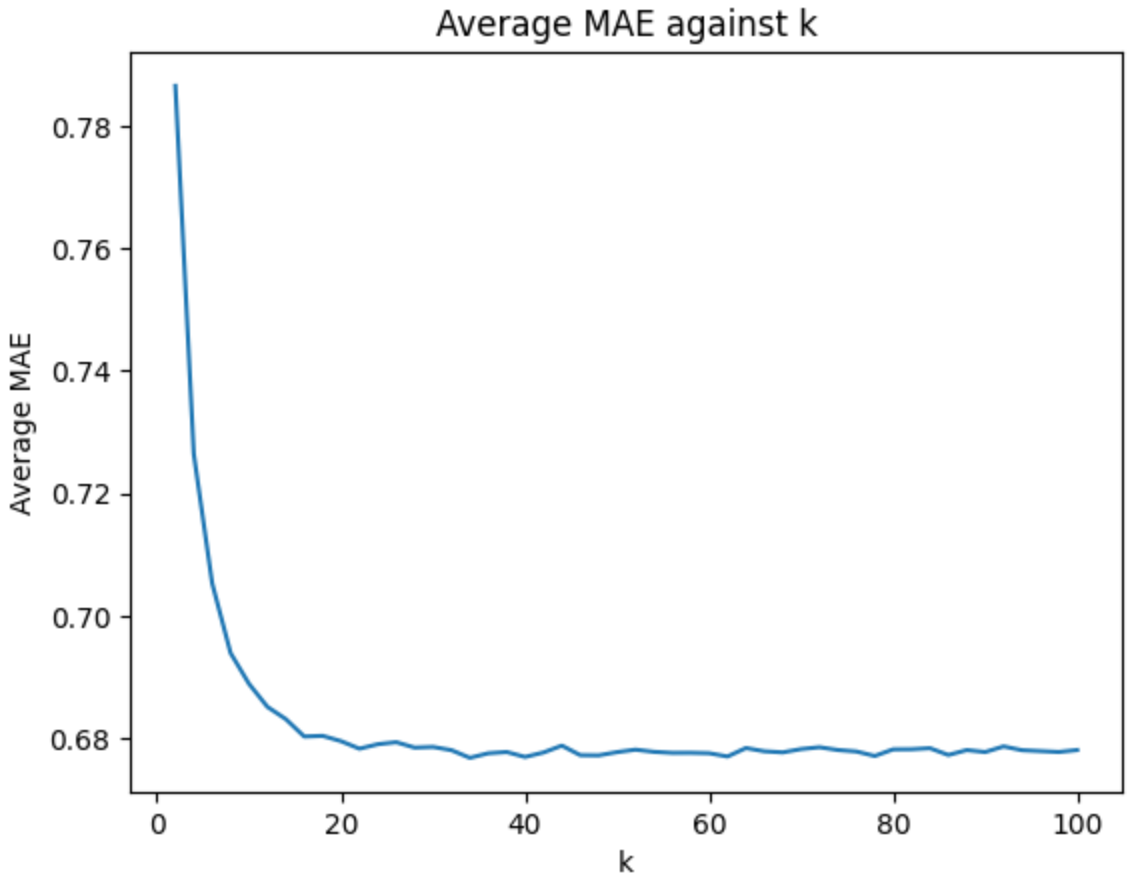
```
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity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

In [12]:

```
plt.plot(ks,avg_rmse)
plt.title("Average RMSE against k" )
plt.ylabel('Average RMSE')
plt.xlabel('k')
plt.show()
```



In [13]:
```
plt.plot(ks,avg_mae)
plt.title("Average MAE against k" )
plt.ylabel('Average MAE')
plt.xlabel('k')
plt.show()
```



Question 5

In [14]:
```
eps = 1e-3
for i in range(len(ks)):
    if((abs(avg_rmse[i]-avg_rmse[i+1])<eps)):
        print(f"Minimum k for RMSE is {ks[i]}, average RMSE is {avg_rmse[i]:.3f}")
        break

for i in range(len(ks)):
    if((abs(avg_mae[i]-avg_mae[i+1])<eps)):
        print(f"Minimum k for MAE is {ks[i]}, average MAE is {avg_mae[i]:.3f}")
        break
```

```
Minimum k for RMSE is 16, average RMSE is 0.893
Minimum k for MAE is 16, average MAE is 0.680
```

Question 6

Question 7 is stated on our report

In [77]:
```
reader = Reader(rating_scale=(0.5, 5))

# data = Dataset.load_from_df(ratings[['userId','movieId','rating']], reader=reader)

data_df = ratings.pivot_table('rating', 'userId', 'movieId')
variance = np.var(data_df, axis=0)

kf = KFold(n_splits=10)

ref = defaultdict(list)
for _, row in ratings.iterrows():
    ref[row['movieId']].append(row['rating'])
```

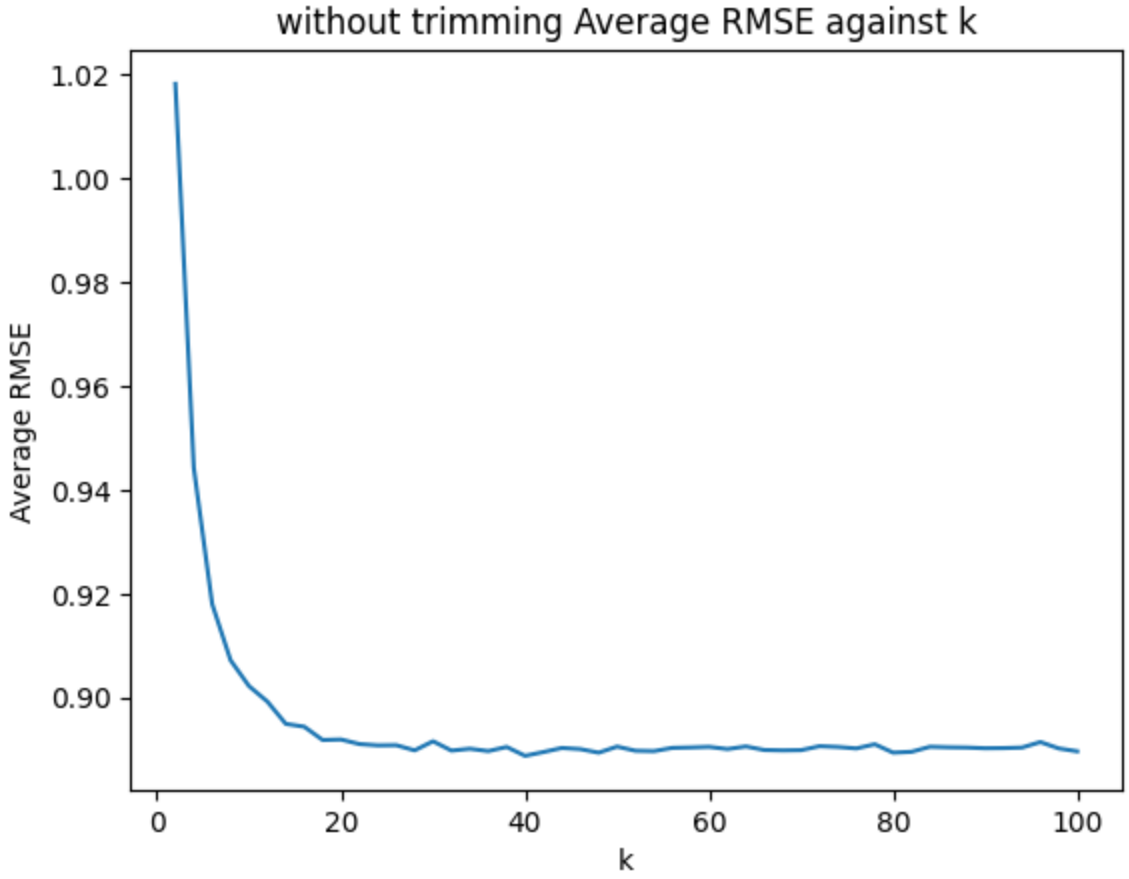
In [78]:
```
def popular_trimming(row):
  return len(ref[row['movieId']]) > 2

def unpopular_trimming(row):
  return len(ref[row['movieId']]) <= 2

def high_variance_trimming(row):
  return len(ref[row['movieId']]) >= 5 and row['movieId'] in variance[variance>=2].keys(

def do_nothing(row):
  return True

def minimum_k(title, avg_rmse, avg_mae, ks):
  eps = 1e-3
  best_k = -1
  for i in range(len(ks)-1):
    if((abs(avg_rmse[i]-avg_rmse[i+1])<eps)):
        print(f"{title} Minimum k for RMSE is {ks[i]}, average RMSE is {avg_rmse[i]:.3f}
        best_k = ks[i]
        break

  for i in range(len(ks)-1):
    if((abs(avg_mae[i]-avg_mae[i+1])<eps)):
        print(f"{title} Minimum k for MAE is {ks[i]}, average MAE is {avg_mae[i]:.3f}")
        break
  return best_k

def plot_k(title, avg_rmse, avg_mae, ks):
    plt.plot(ks,avg_rmse)
    plt.title(f"{title} Average RMSE against k" )
```

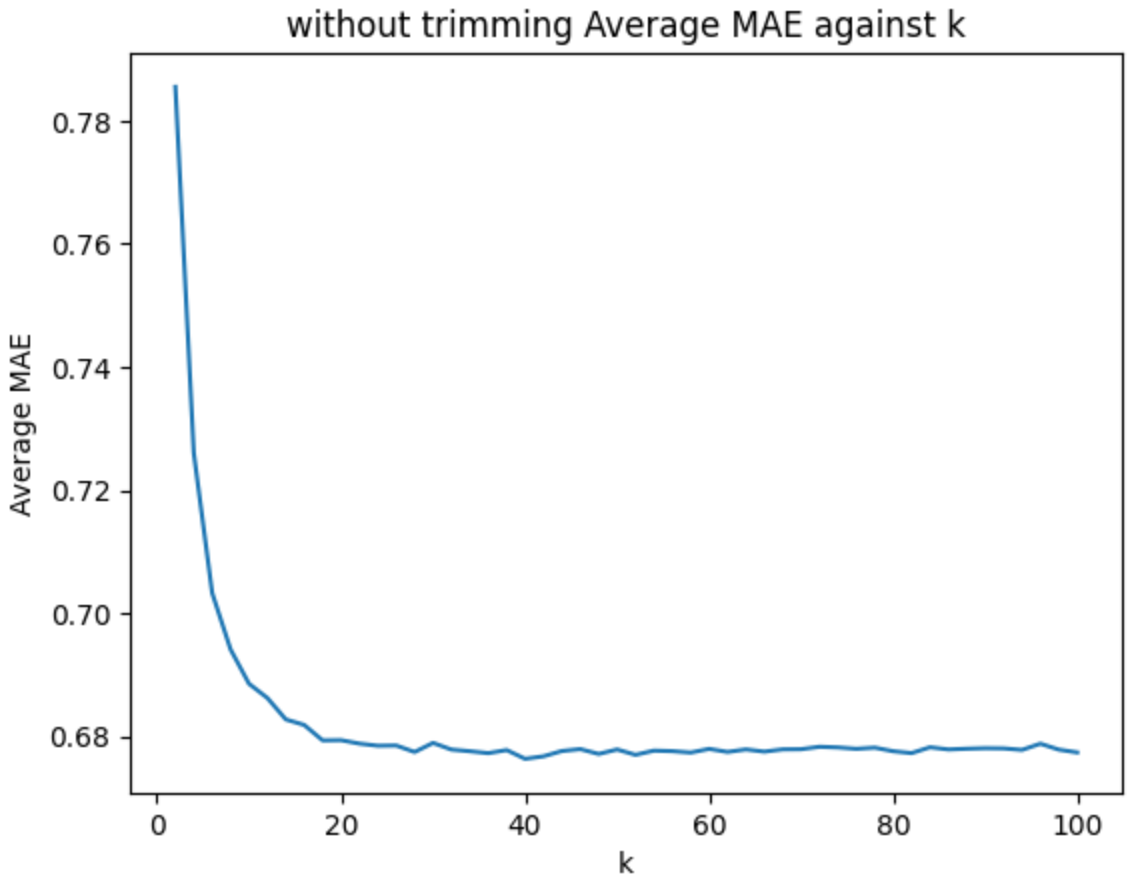
```
    plt.ylabel('Average RMSE')
    plt.xlabel('k')
    plt.show()

    plt.plot(ks,avg_mae)
    plt.title(f"{title} Average MAE against k" )
    plt.ylabel('Average MAE')
    plt.xlabel('k')
    plt.show()

def report_roc(trainset, testset, best_k, title):
  Threshold_list = [2.5, 3.0, 3.5, 4.0]
  res  = KNNWithMeans(k=best_k,sim_options={'name':'pearson'},verbose=False).fit(trainse

  for thre in Threshold_list:
    thresholded_out = []
    for row in res:
        if row.r_ui > thre:
            thresholded_out.append(1)
        else:
            thresholded_out.append(0)
    FPR, TPR, thresholds = roc_curve(thresholded_out, [row.est for row in res])
    labels = f"AUC: {auc(FPR,TPR):.3f} , threshold: {thre}"
    plt.plot(FPR, TPR,lw=2, label=labels)

  plt.legend(loc='best')
  plt.title(f'{title} ROC of best_k is {best_k}')
  plt.ylabel('True Positive Rate- TPR')
  plt.xlabel('False Positive Rate- FPR')
  plt.show()

def report_result(ratings, filter, title, ks):
  filter_ratings = ratings[ratings.apply(filter, axis=1)]
  data = Dataset.load_from_df(filter_ratings[['userId','movieId','rating']], reader=read

  avg_rmse = []
  avg_mae = []
  rmse_min = 1e3
  rmse_min_k = ks[-1]
  mae_min = 1e3
  mae_min_k = ks[-1]

  for k in ks:
      print(f"k={k}", end=":")
      rmse = 0
      mae = 0
      iter = 1
      for trainset, testset in kf.split(data):
          print(f"iter={iter}", end=",")
          iter += 1
          perf = KNNWithMeans(k=k,sim_options={'name':'pearson'},verbose=False).fit(trai
          rmse += accuracy.rmse(perf,verbose=False)
          mae += accuracy.mae(perf,verbose=False)
      print("")
      if rmse < rmse_min:
        rmse_min = rmse
        rmse_min_k = k
      if mae < mae_min:
        mae_min = mae
        mae_min_k = k
      avg_rmse.append(rmse / 10.0)
      avg_mae.append(mae / 10.0)

  plot_k(title, avg_rmse, avg_mae,ks)
  best_k = minimum_k(title, avg_rmse, avg_mae,ks)
  if best_k == -1:
```

```
    print("no convergence, we just choose a k with min rmse")
    best_k = rmse_min_k
  report_roc(trainset, testset, best_k,title)
```

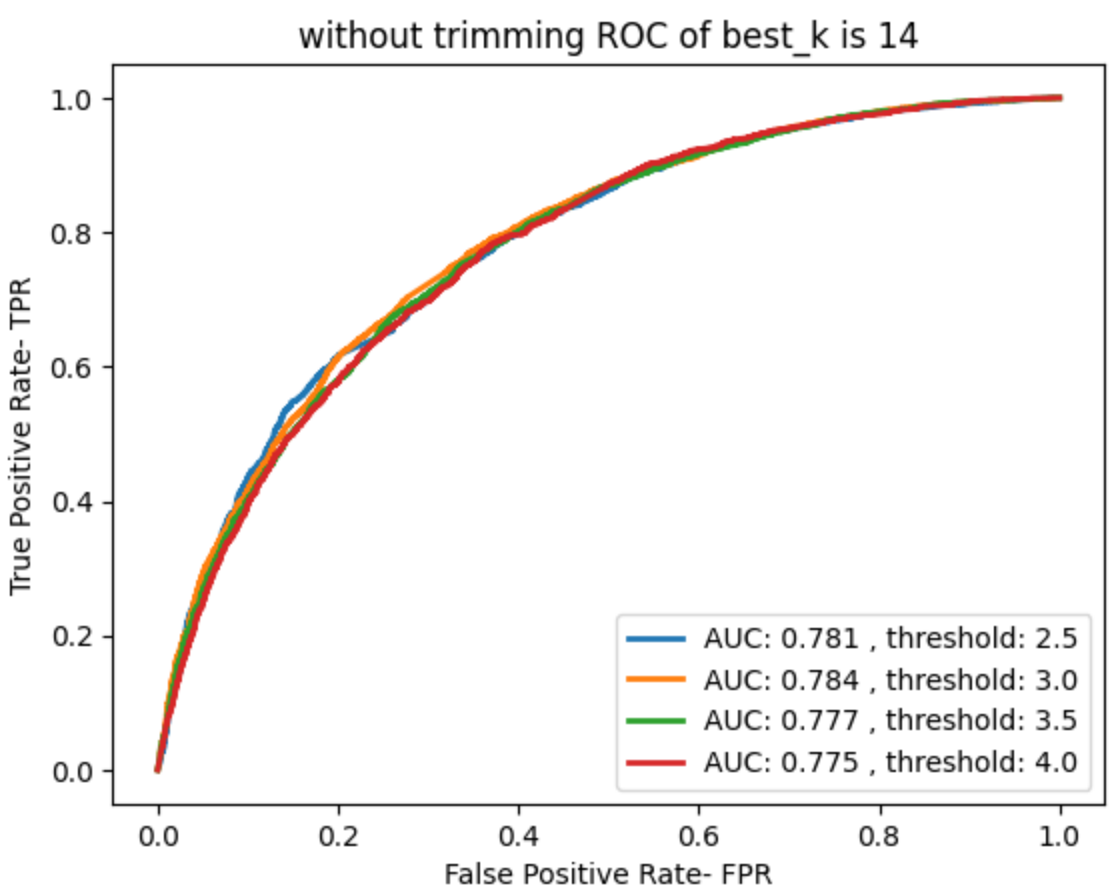
In [79]:
```
report_result(ratings, do_nothing, "without trimming", ks=np.arange(2,102,2))
```

```
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=26:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=28:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=30:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=32:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=34:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=36:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=38:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=40:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=42:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=44:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=46:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=48:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=50:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=52:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=54:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=56:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=58:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=60:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=62:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=64:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=66:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=68:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=70:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=72:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=74:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=76:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=78:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=80:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=82:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=84:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=86:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=88:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=90:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=92:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=94:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=96:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=98:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=100:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```





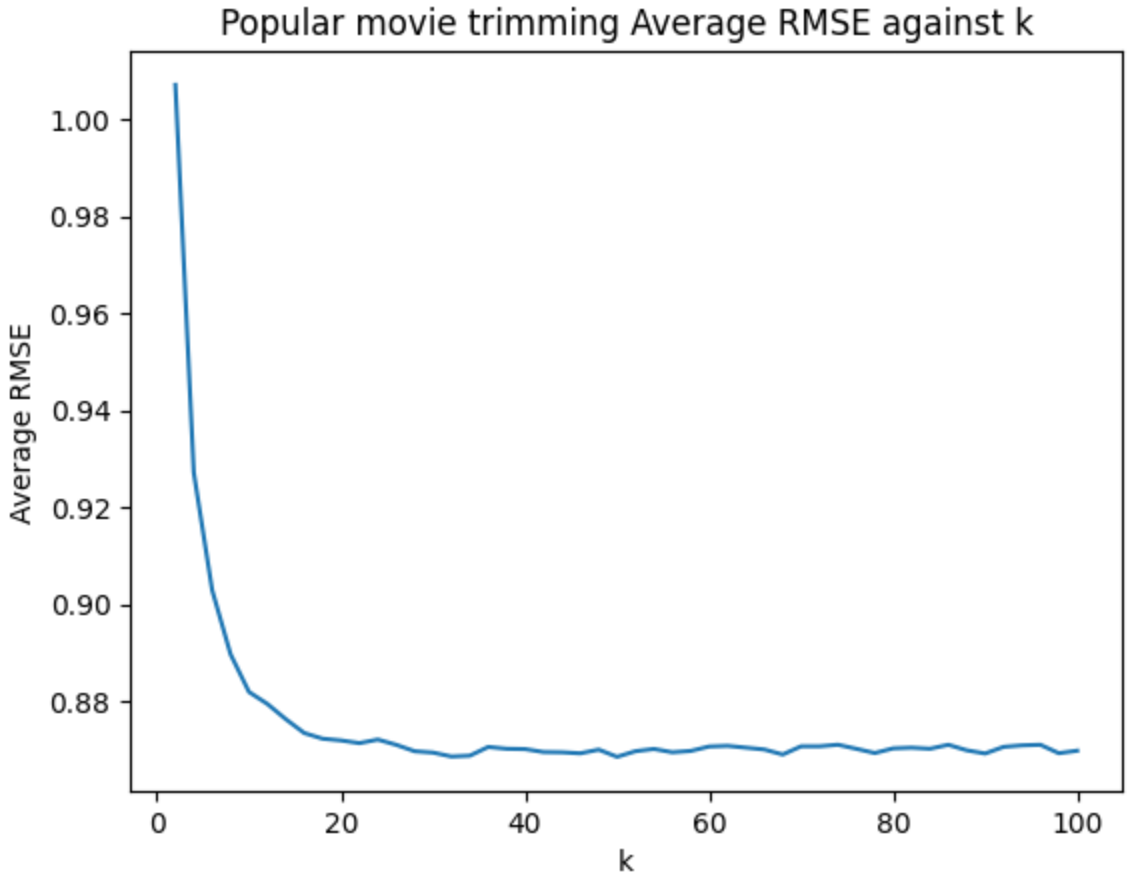

```
without trimming Minimum k for RMSE is 14, average RMSE is 0.895
without trimming Minimum k for MAE is 14, average MAE is 0.683
```

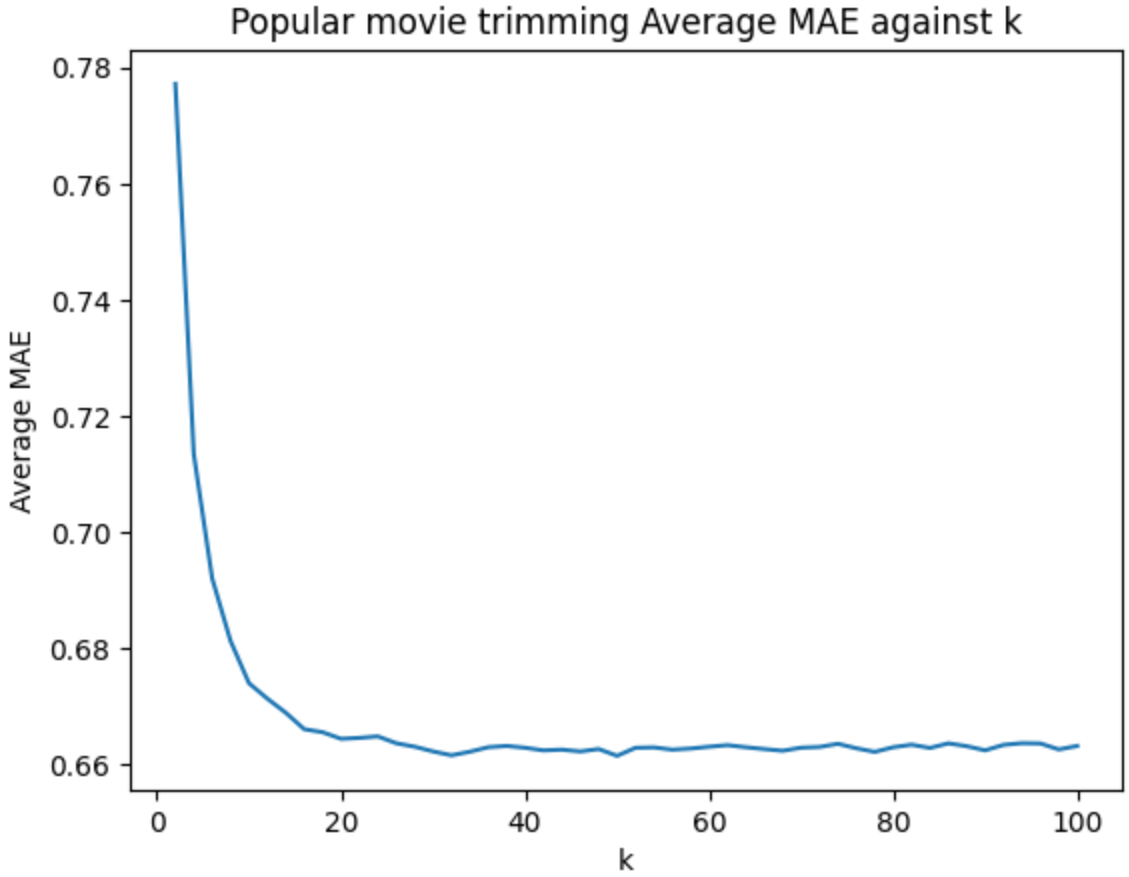


In [80]:
```
report_result(ratings, popular_trimming, "Popular movie trimming", ks=np.arange(2,102,2)
```

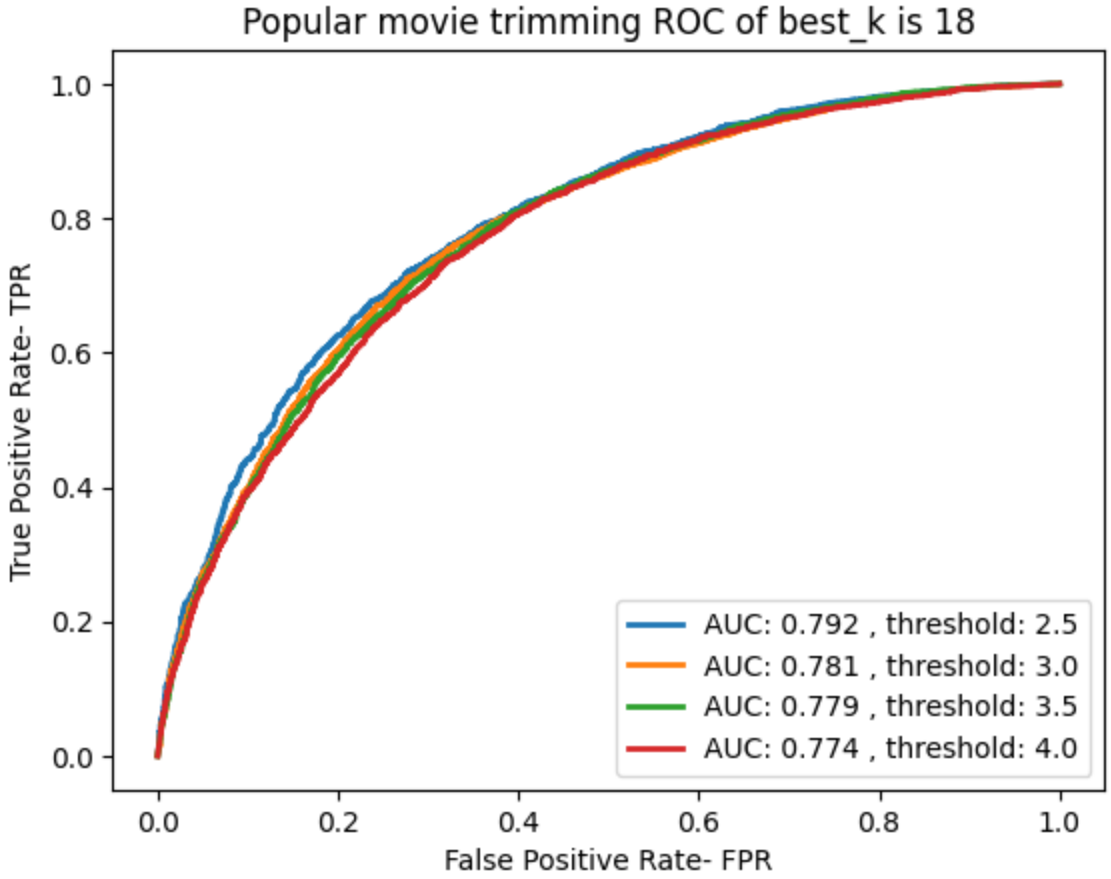
```
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=26:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=28:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=30:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=32:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=34:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=36:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=38:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=40:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=42:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=44:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=46:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=48:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=50:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=52:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=54:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=56:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=58:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=60:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=62:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=64:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=66:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=68:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```

k=70:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=72:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=74:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=76:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=78:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=80:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=82:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=84:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=86:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=88:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=90:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=92:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=94:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=96:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=98:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=100:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,





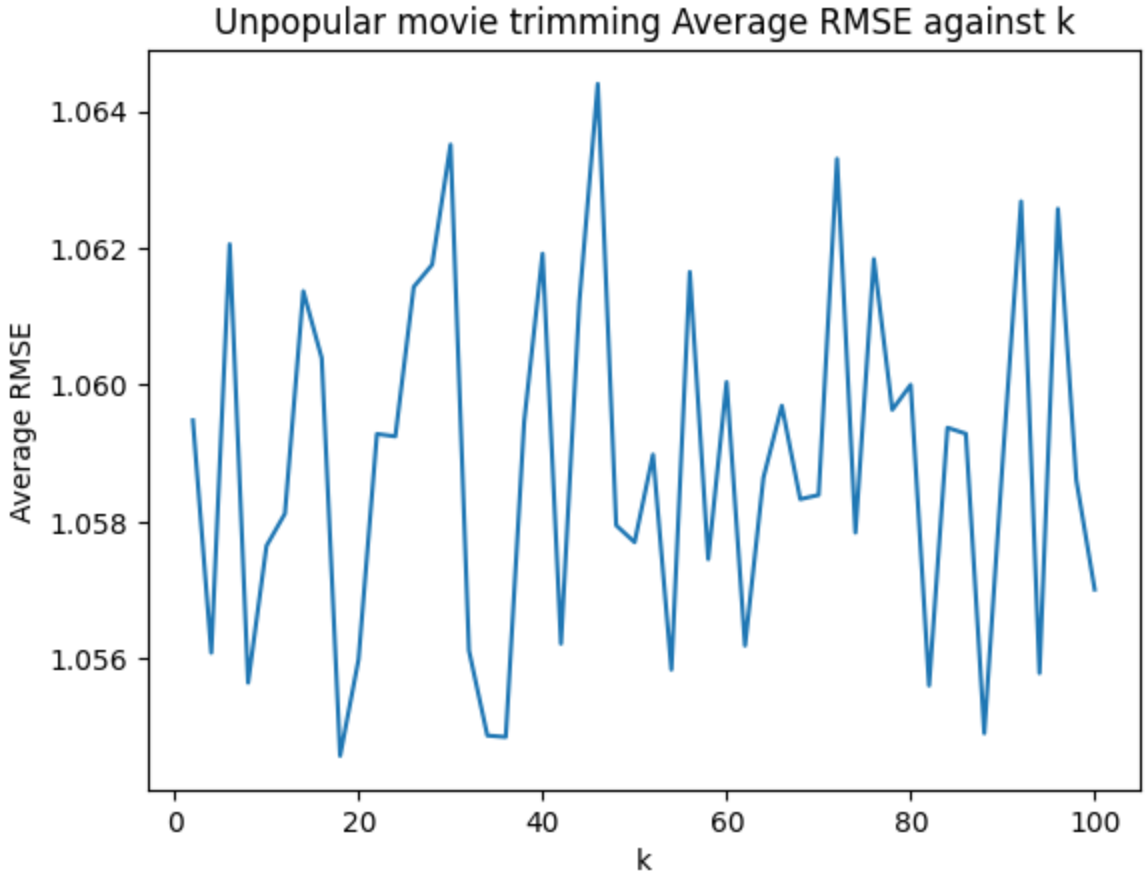

```
Popular movie trimming Minimum k for RMSE is 18, average RMSE is 0.872
Popular movie trimming Minimum k for MAE is 16, average MAE is 0.666
```

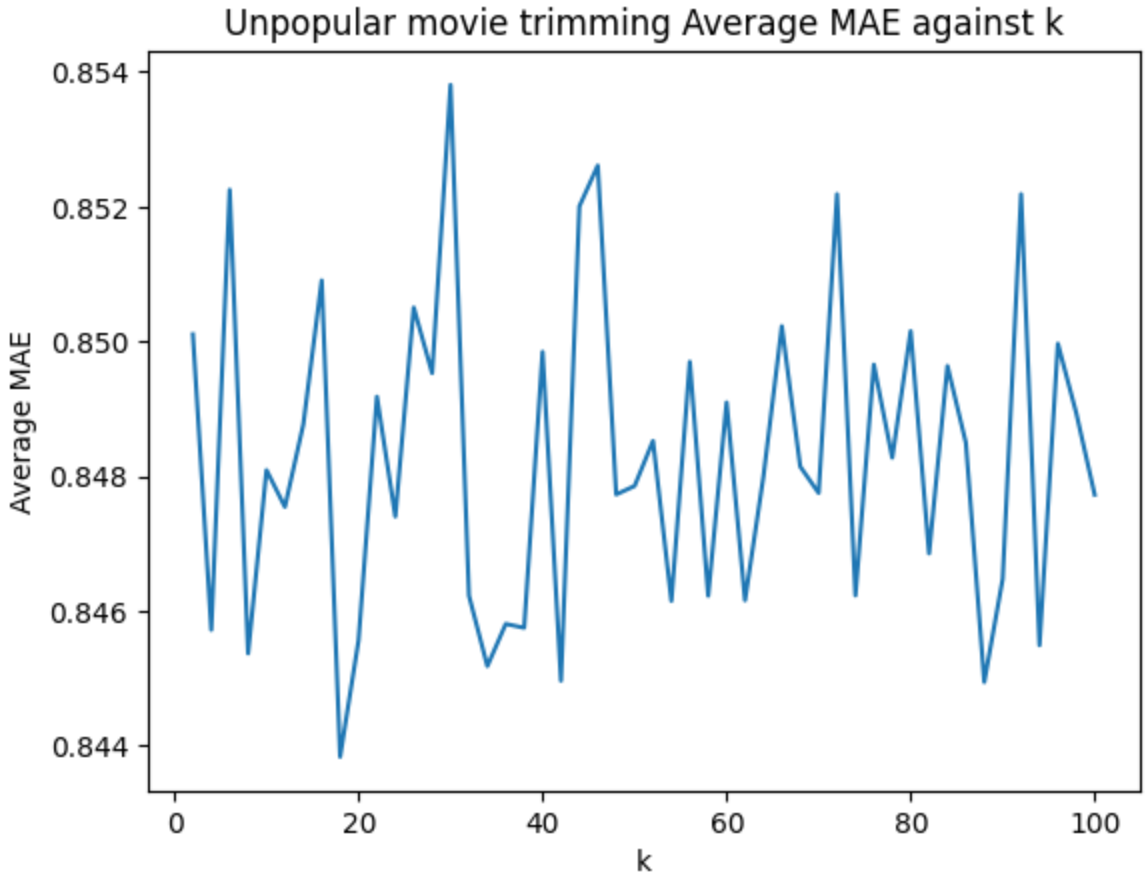


In [81]:
```
report_result(ratings, unpopular_trimming, "Unpopular movie trimming", ks=np.arange(2,10
```

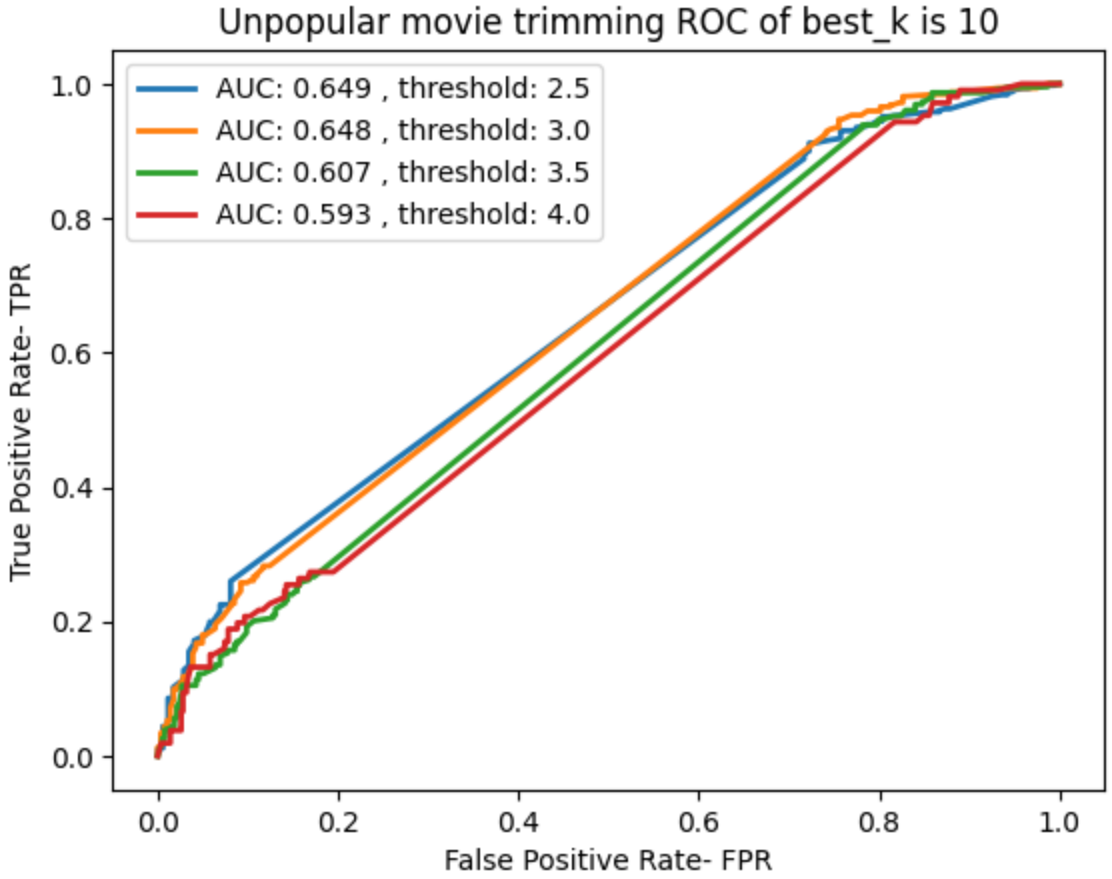
```
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```

k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=26:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=28:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=30:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=32:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=34:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=36:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=38:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=40:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=42:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=44:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=46:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=48:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=50:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=52:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=54:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=56:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=58:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=60:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=62:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=64:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=66:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=68:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=70:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=72:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=74:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=76:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=78:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=80:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=82:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=84:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=86:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=88:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=90:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=92:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=94:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=96:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=98:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=100:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,





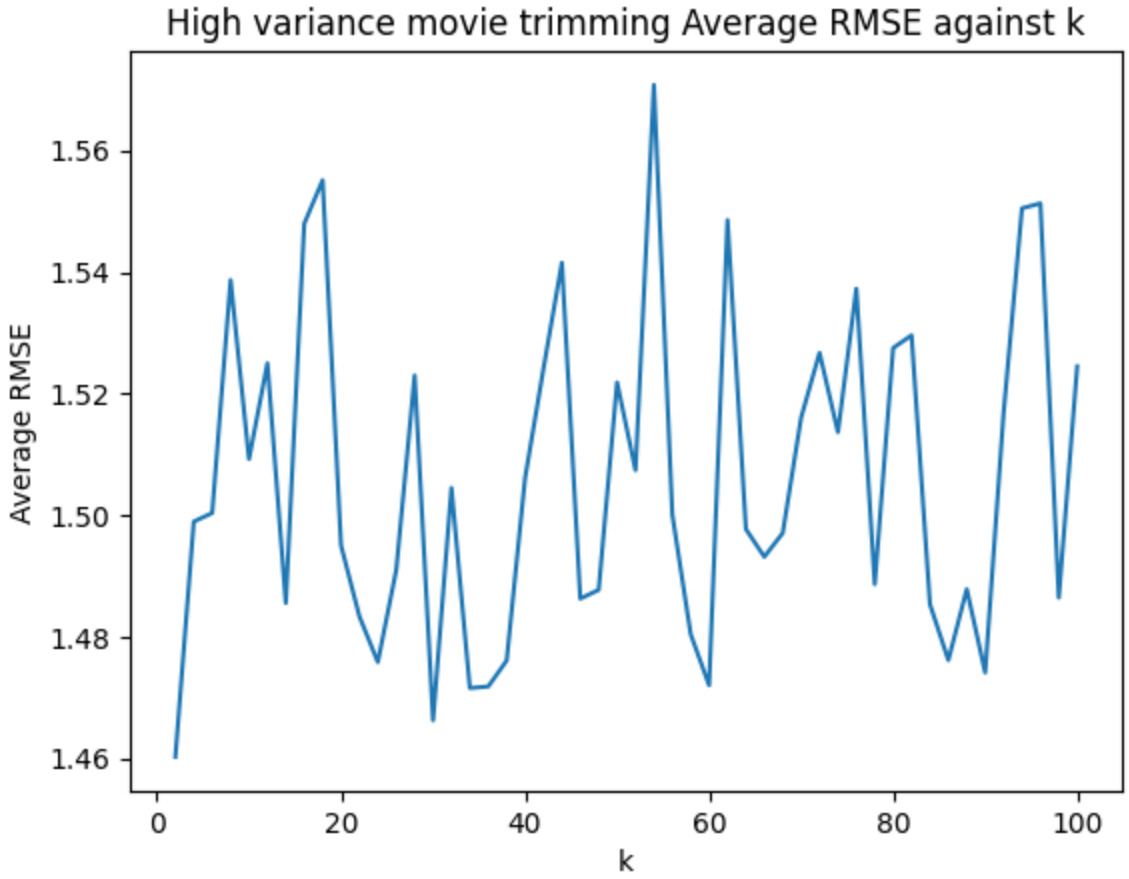

```
Unpopular movie trimming Minimum k for RMSE is 10, average RMSE is 1.058
Unpopular movie trimming Minimum k for MAE is 10, average MAE is 0.848
```

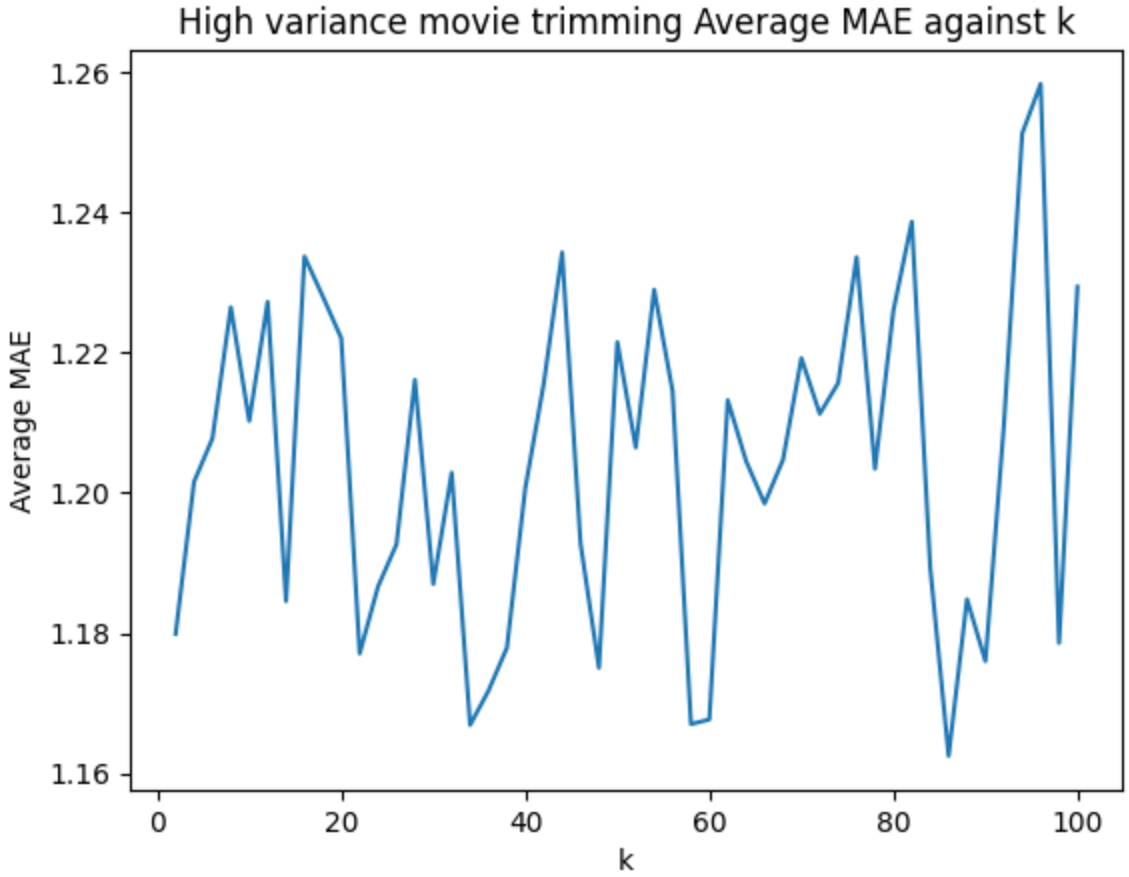


In [102…
```
report_result(ratings, high_variance_trimming, "High variance movie trimming", ks=np.ara
```

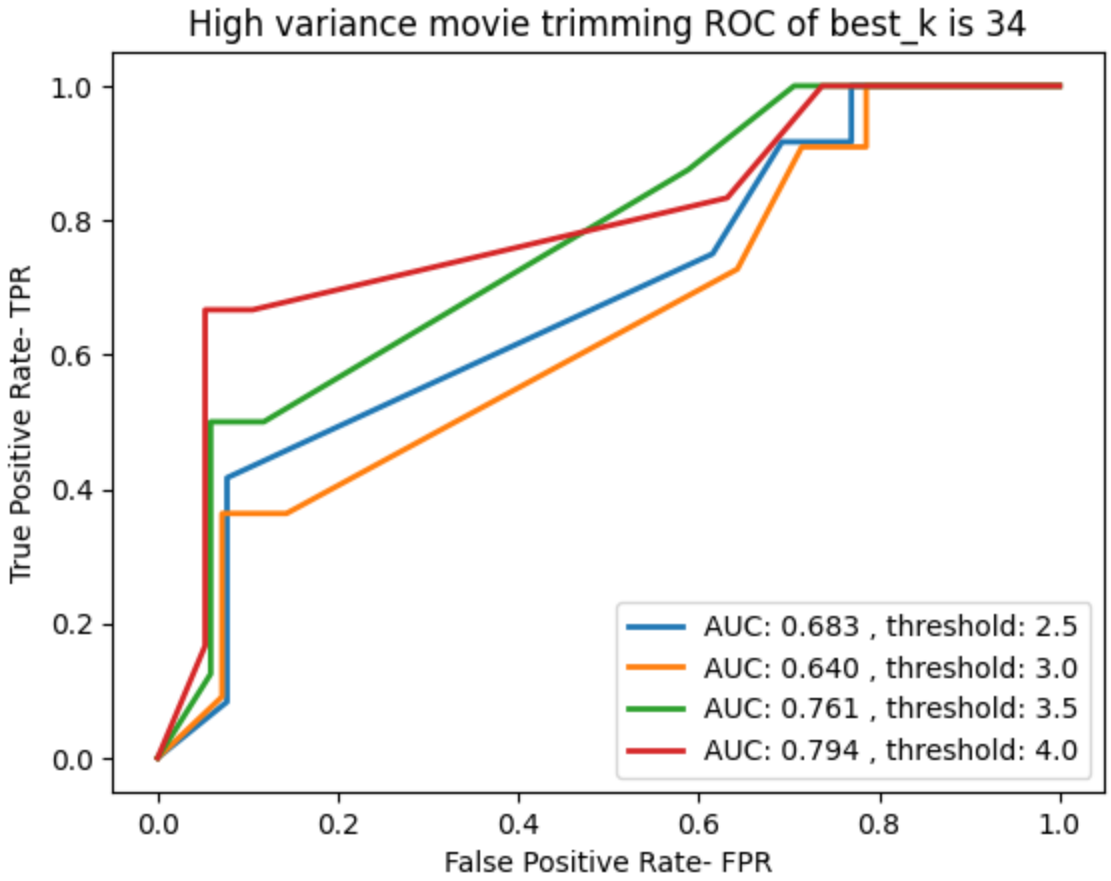
```
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=26:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=28:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=30:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=32:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=34:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=36:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=38:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=40:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=42:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=44:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=46:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=48:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=50:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=52:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=54:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=56:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=58:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=60:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=62:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=64:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=66:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=68:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```

k=70:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=72:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=74:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=76:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=78:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=80:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=82:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=84:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=86:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=88:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=90:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=92:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=94:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=96:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=98:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=100:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,






```
High variance movie trimming Minimum k for RMSE is 34, average RMSE is 1.472
High variance movie trimming Minimum k for MAE is 58, average MAE is 1.167
```



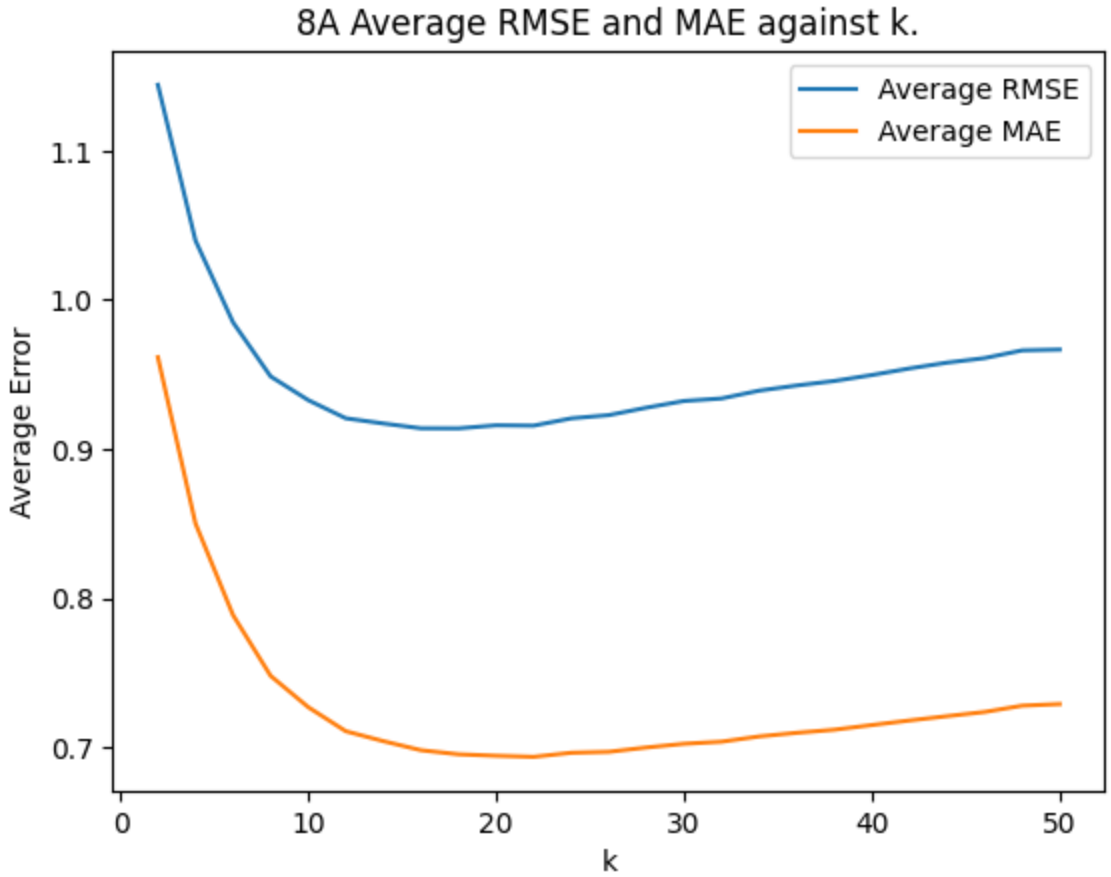
Question 8

In [4]:
```
avg_rmse = []
avg_mae = []
```

```
min_rmse = 1e4
min_mae = 1e4
min_rmse_k = 0
min_rmsse_k = 0

new_ks = np.arange(2,52,2)
for i in trange(len(new_ks)):
  k = new_ks[i]
  print(k)
  nmf = NMF(n_factors = k)
  cv_ = cross_validate(nmf, data, measures=['rmse', 'mae'], cv=10, verbose=False)

  rmse = np.mean(cv_["test_rmse"])
  avg_rmse.append(rmse)
  if rmse < min_rmse:
    min_rmse = rmse
    min_rmse_k = k

  mae = np.mean(cv_["test_mae"])
  avg_mae.append(mae)
  if mae < min_mae:
    min_mae = mae
    min_mae_k = k

plt.plot(new_ks, avg_rmse, label="Average RMSE")
plt.plot(new_ks, avg_mae, label="Average MAE")
plt.title("8A NMF Average RMSE and MAE against k.")
plt.legend()
plt.xlabel("k")
plt.ylabel("Average Error")
plt.show()
```

```
  0%|          | 0/25 [00:00<?, ?it/s]
2
  4%|▍         | 1/25 [00:14<05:50, 14.61s/it]
4
  8%|▊         | 2/25 [00:29<05:37, 14.65s/it]
6
 12%|█▏        | 3/25 [00:44<05:29, 14.97s/it]
8
 16%|█▌        | 4/25 [01:01<05:27, 15.61s/it]
10
 20%|██        | 5/25 [01:21<05:43, 17.20s/it]
12
 24%|██▍       | 6/25 [01:38<05:28, 17.31s/it]
14
 28%|██▊       | 7/25 [02:00<05:35, 18.63s/it]
16
 32%|███▏      | 8/25 [02:18<05:16, 18.63s/it]
18
 36%|███▌      | 9/25 [02:37<04:59, 18.74s/it]
20
 40%|████      | 10/25 [02:57<04:47, 19.14s/it]
22
 44%|████▍     | 11/25 [03:17<04:31, 19.41s/it]
24
 48%|████▊     | 12/25 [03:38<04:18, 19.85s/it]
26
 52%|█████▏    | 13/25 [04:02<04:13, 21.13s/it]
28
```

```
 56%|█████▌    | 14/25 [04:24<03:54, 21.35s/it]
30
 60%|██████    | 15/25 [04:46<03:36, 21.62s/it]
32
 64%|██████▍   | 16/25 [05:10<03:19, 22.13s/it]
34
 68%|██████▊   | 17/25 [05:35<03:03, 22.99s/it]
36
 72%|███████▏  | 18/25 [05:59<02:44, 23.51s/it]
38
 76%|███████▌  | 19/25 [06:26<02:26, 24.44s/it]
40
 80%|████████  | 20/25 [06:52<02:04, 24.92s/it]
42
 84%|████████▍ | 21/25 [07:17<01:39, 24.92s/it]
44
 88%|████████▊ | 22/25 [07:42<01:15, 25.04s/it]
46
 92%|█████████▏| 23/25 [08:17<00:55, 27.82s/it]
48
 96%|█████████▌| 24/25 [08:44<00:27, 27.66s/it]
50
100%|██████████| 25/25 [09:17<00:00, 22.30s/it]
```



In [5]:
```
print("Question8B")
print(f"Minimum Average RMSE (NMF) = {min_rmse:.3f}, k = {min_rmse_k}")
print(f"Minimum Average MAE (NMF) = {min_mae:.3f}, k = {min_mae_k}")
```

```
Question8B
Minimum Average RMSE (NMF) = 0.914, k = 18
Minimum Average MAE (NMF) = 0.694, k = 22
```

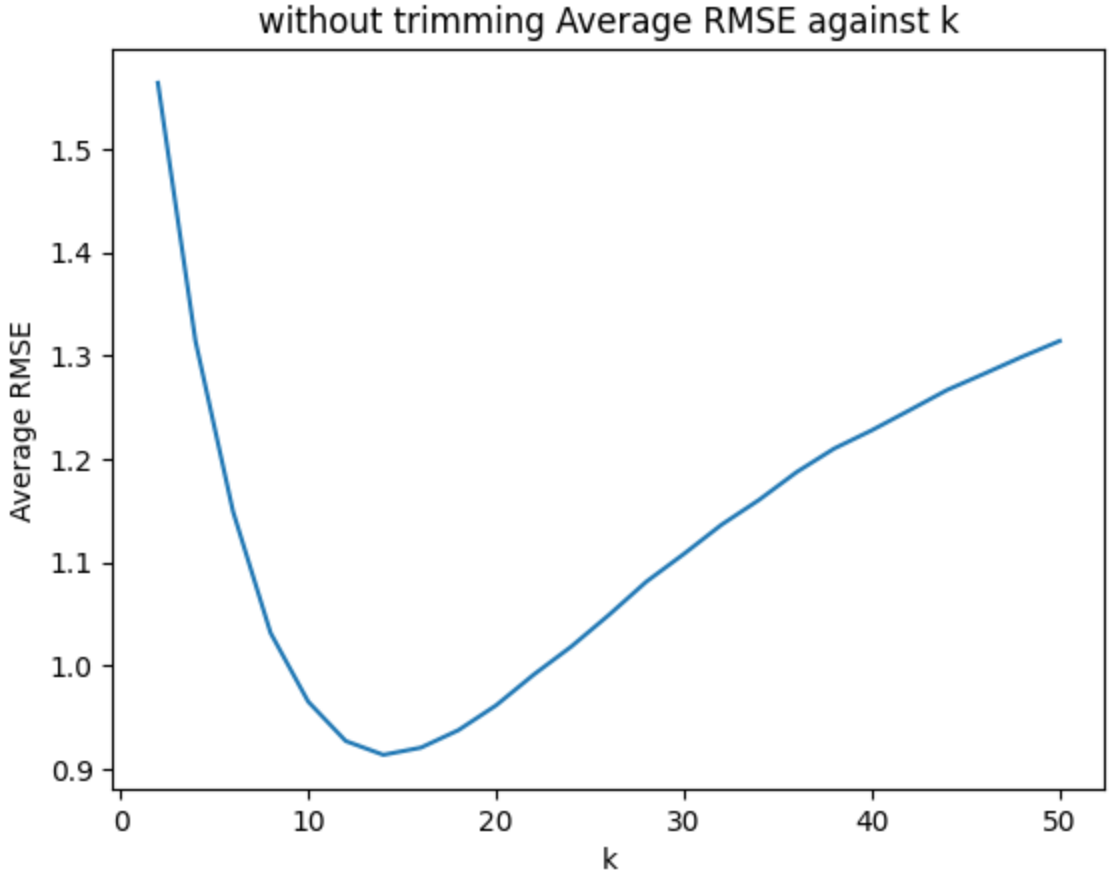
In [84]:
```
def report_roc_nmf(trainset, testset, best_k, title):
  Threshold_list = [2.5, 3.0, 3.5, 4.0]
```

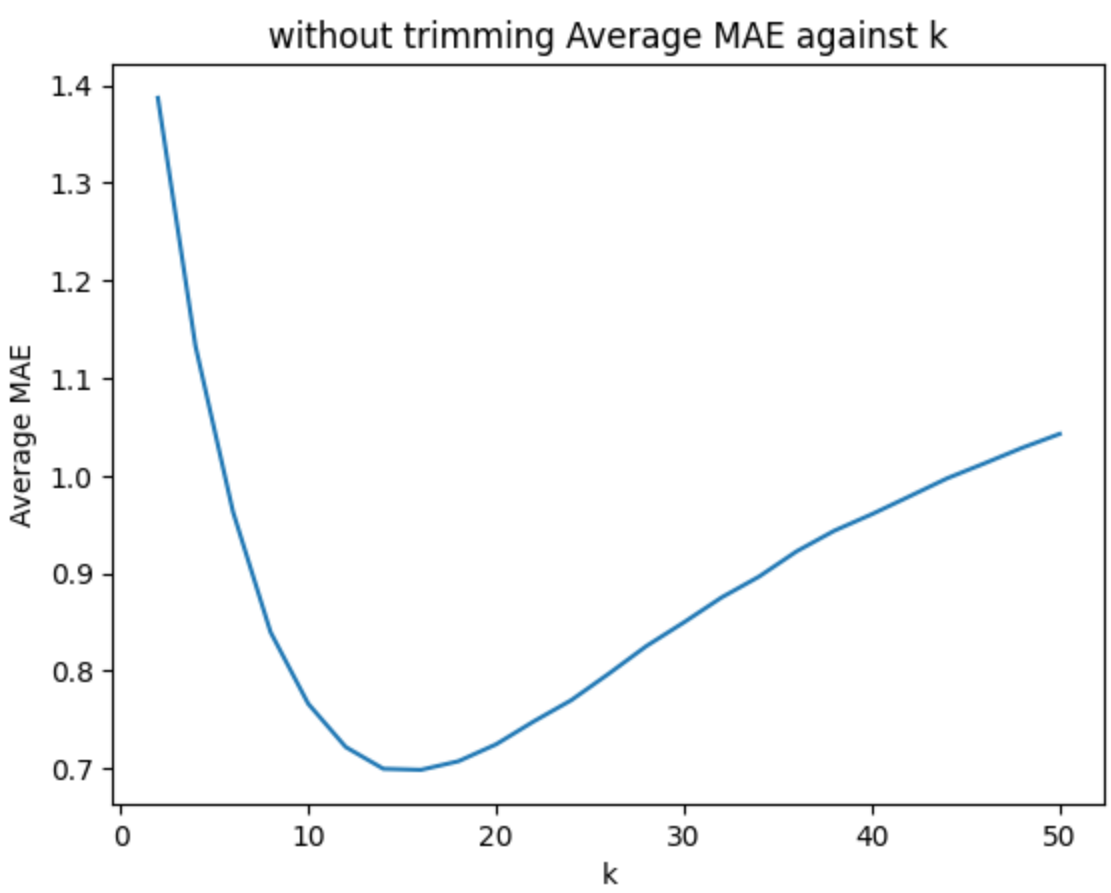
```
  res  = NMF(n_factors=best_k,n_epochs=30,verbose=False).fit(trainset).test(testset)

  for thre in Threshold_list:
    thresholded_out = []
    for row in res:
        if row.r_ui > thre:
            thresholded_out.append(1)
        else:
            thresholded_out.append(0)
    FPR, TPR, thresholds = roc_curve(thresholded_out, [row.est for row in res])
    labels = f"AUC: {auc(FPR,TPR):.3f} , threshold: {thre}"
    plt.plot(FPR, TPR,lw=2, label=labels)

  plt.legend(loc='best')
  plt.title(f'{title} ROC of best_k is {best_k}')
  plt.ylabel('True Positive Rate- TPR')
  plt.xlabel('False Positive Rate- FPR')
  plt.show()

def report_result_nmf(ratings, filter, title, ks):
  filter_ratings = ratings[ratings.apply(filter, axis=1)]
  data = Dataset.load_from_df(filter_ratings[['userId','movieId','rating']], reader=read

  avg_rmse = []
  avg_mae = []
  rmse_min = 1e3
  rmse_min_k = ks[-1]
  mae_min = 1e3
  mae_min_k = ks[-1]

  for k in ks:
      print(f"k={k}", end=":")
      rmse = 0
      mae = 0
      iter = 1
      for trainset, testset in kf.split(data):
          print(f"iter={iter}", end=",")
          iter += 1
          perf = NMF(n_factors=k,n_epochs=30,verbose=False).fit(trainset).test(testset)
          rmse += accuracy.rmse(perf,verbose=False)
          mae += accuracy.mae(perf,verbose=False)
      print("")
      if rmse < rmse_min:
        rmse_min = rmse
        rmse_min_k = k
      if mae < mae_min:
        mae_min = mae
        mae_min_k = k
      avg_rmse.append(rmse / 10.0)
      avg_mae.append(mae / 10.0)

  plot_k(title, avg_rmse, avg_mae,ks)
  best_k = minimum_k(title, avg_rmse, avg_mae,ks)
  if best_k == -1:
    print("no convergence, we just choose a k with min rmse")
    best_k = rmse_min_k
  report_roc_nmf(trainset, testset, best_k,title)
```

In [85]:
```
report_result_nmf(ratings, do_nothing, "without trimming", ks=np.arange(2,52,2))
```

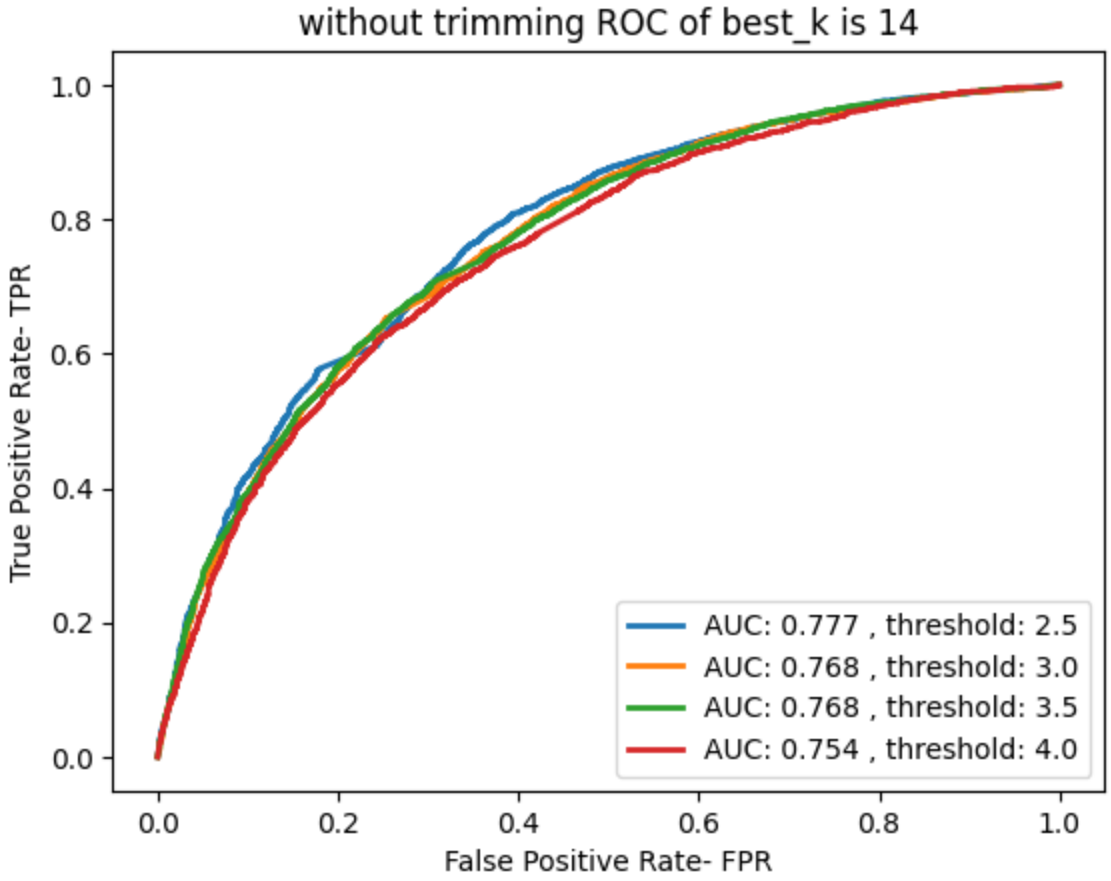
```
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```

k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=26:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=28:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=30:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=32:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=34:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=36:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=38:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=40:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=42:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=44:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=46:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=48:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=50:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,





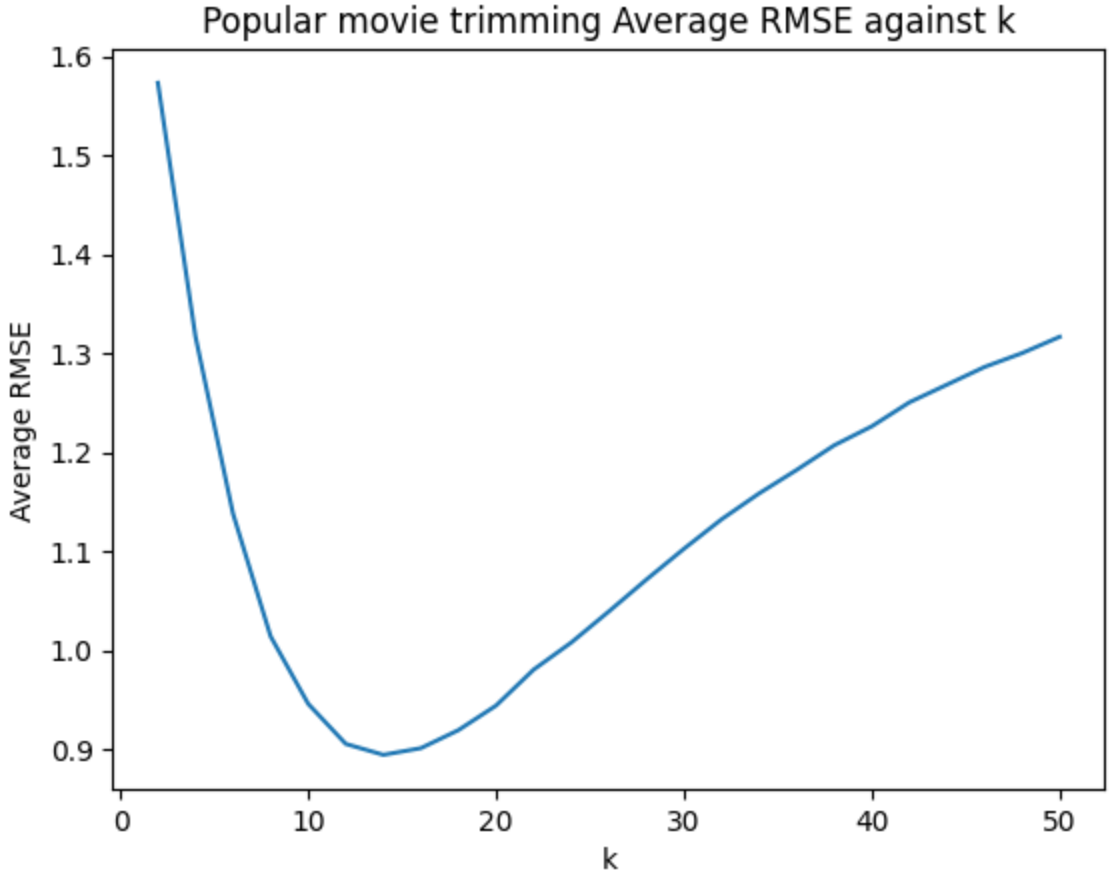

```
no convergence, we just choose a k with min rmse
```

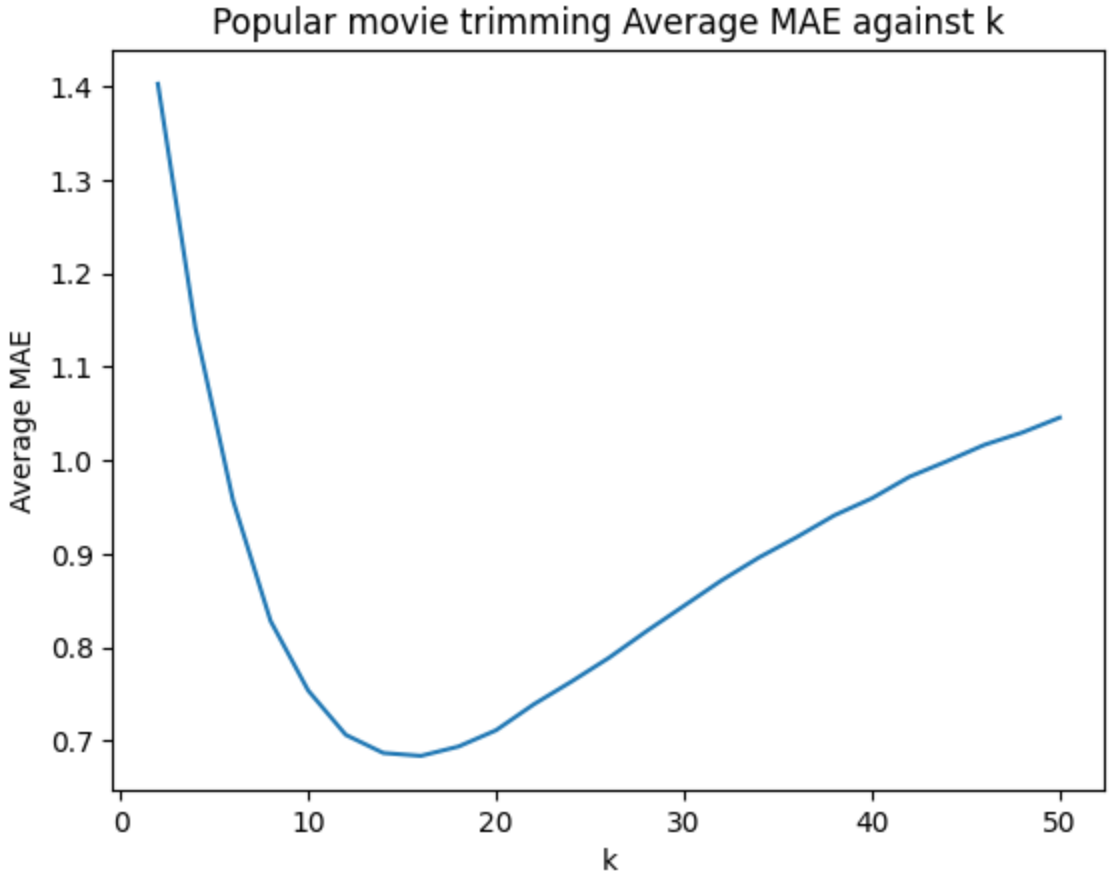


In [86]:
```
report_result_nmf(ratings, popular_trimming, "Popular movie trimming", ks=np.arange(2,52
```

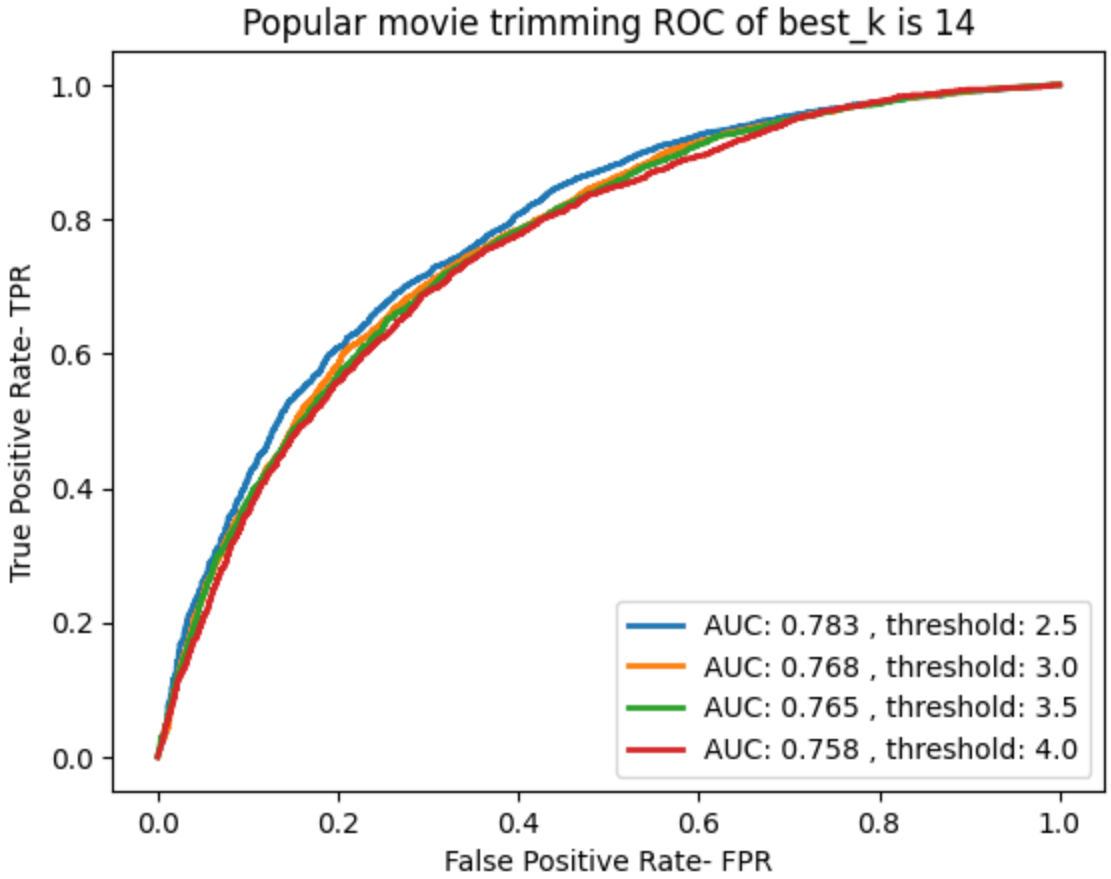
```
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```

k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=26:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=28:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=30:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=32:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=34:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=36:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=38:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=40:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=42:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=44:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=46:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=48:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=50:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,





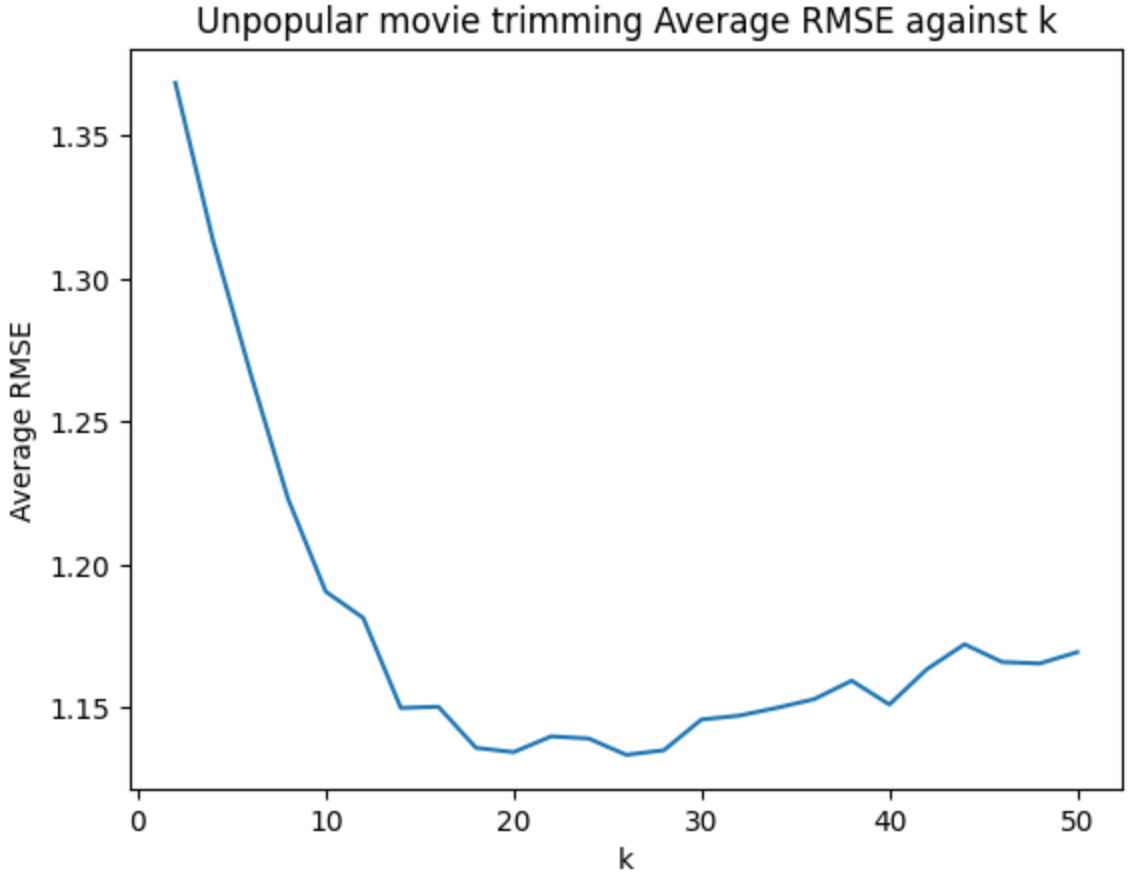

```
no convergence, we just choose a k with min rmse
```

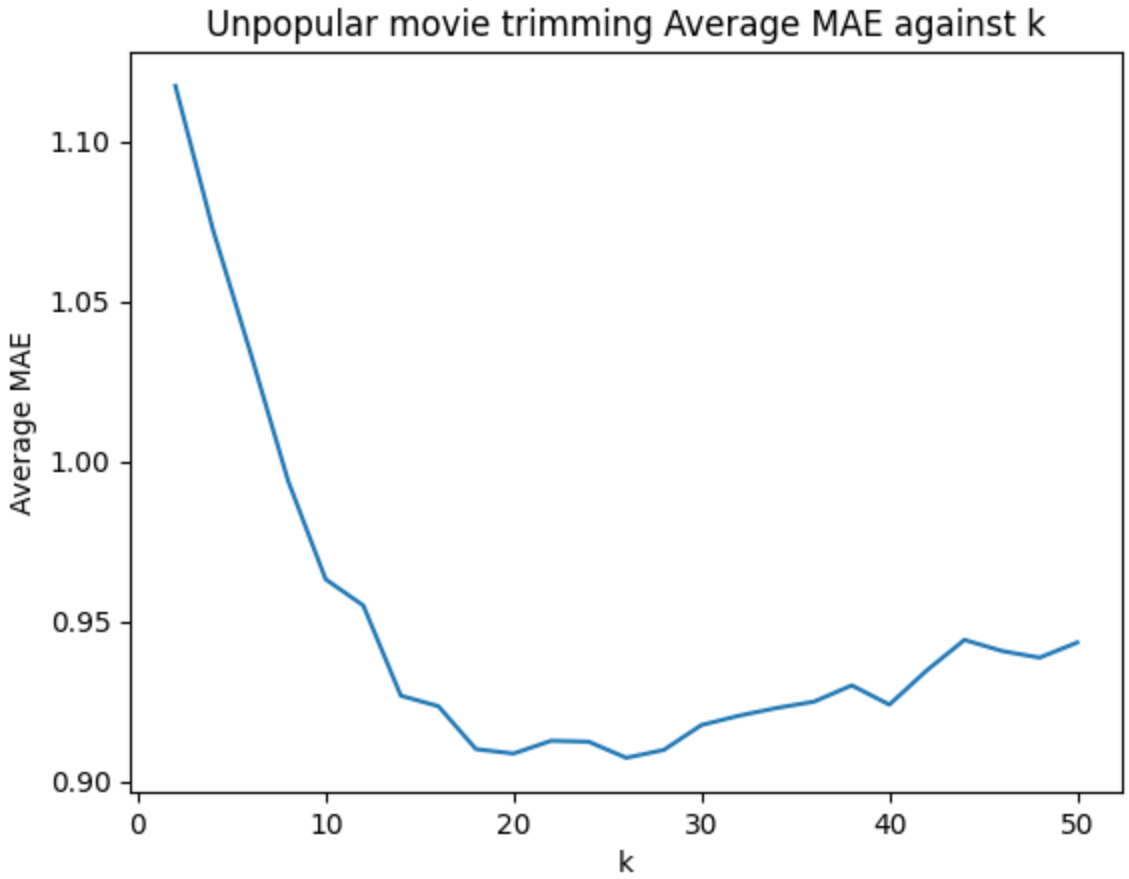


In [87]:
```
report_result_nmf(ratings, unpopular_trimming, "Unpopular movie trimming", ks=np.arange(
```

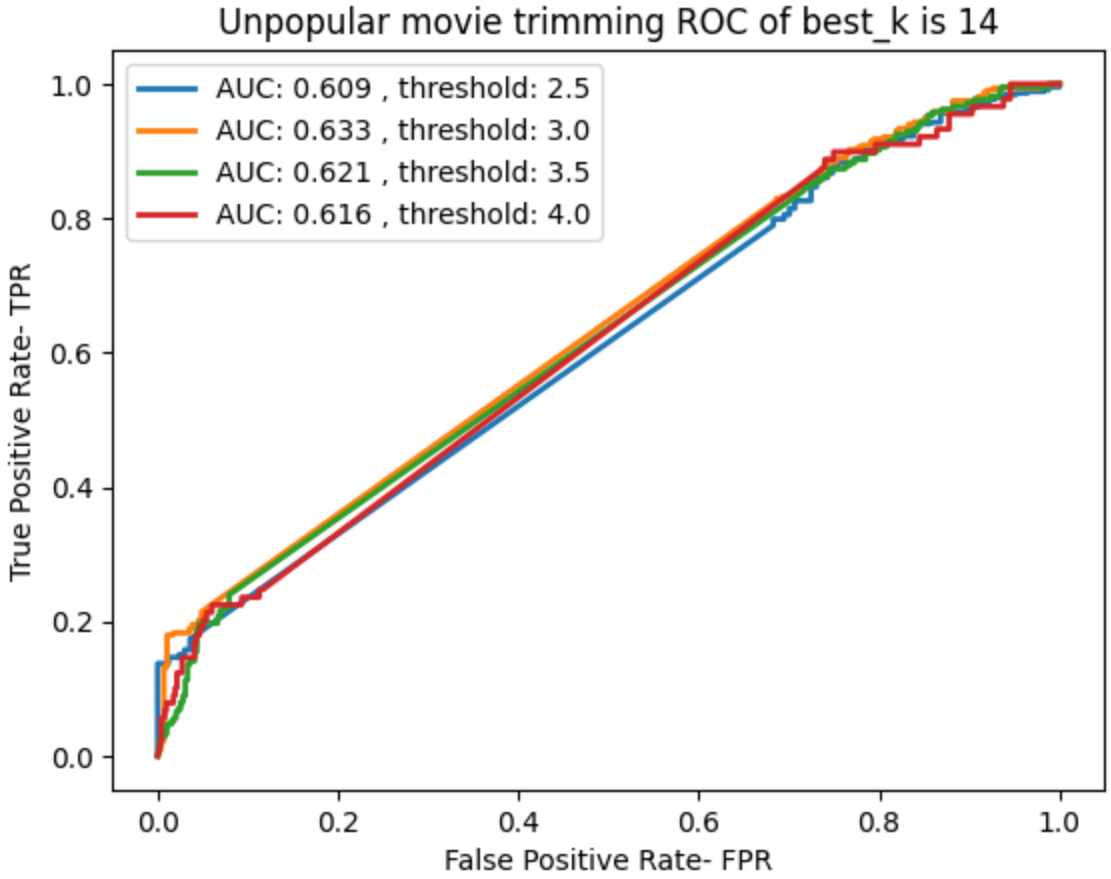
```
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```

k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=26:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=28:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=30:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=32:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=34:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=36:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=38:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=40:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=42:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=44:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=46:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=48:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=50:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,





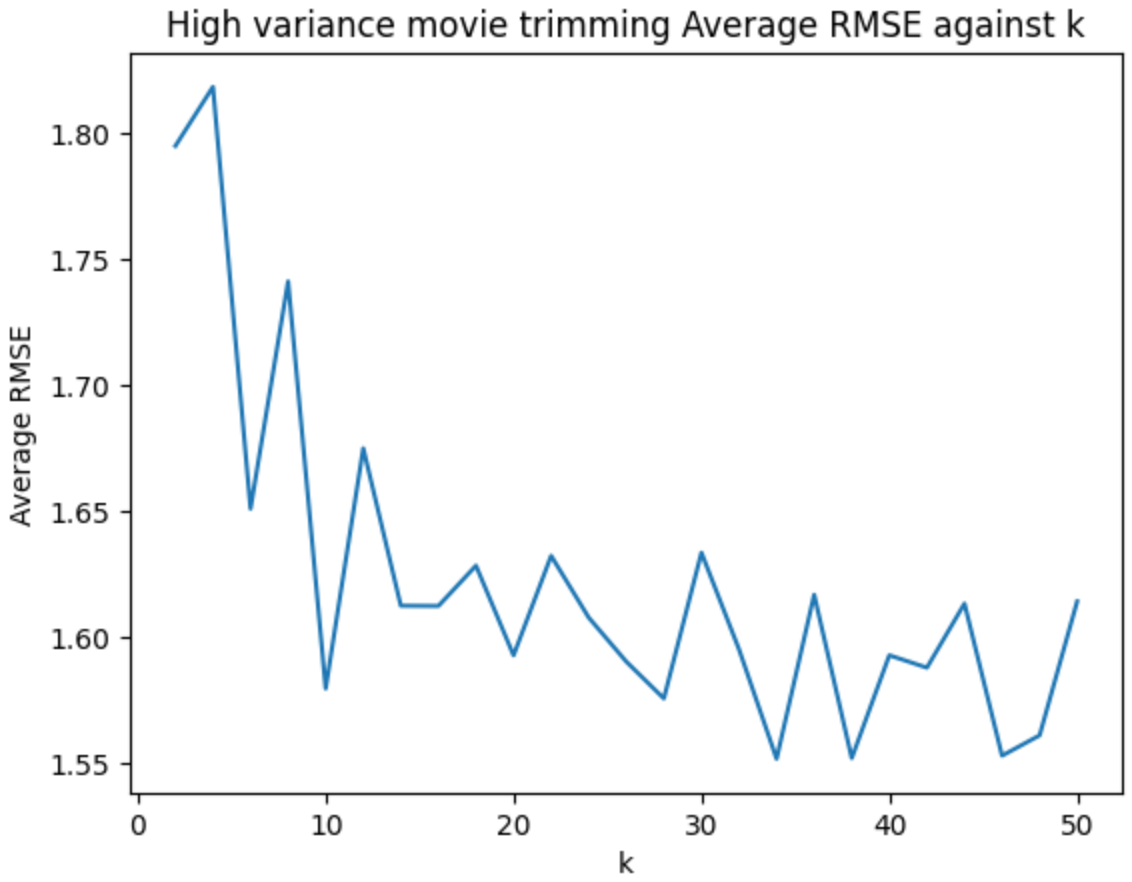

```
Unpopular movie trimming Minimum k for RMSE is 14, average RMSE is 1.150
Unpopular movie trimming Minimum k for MAE is 22, average MAE is 0.913
```

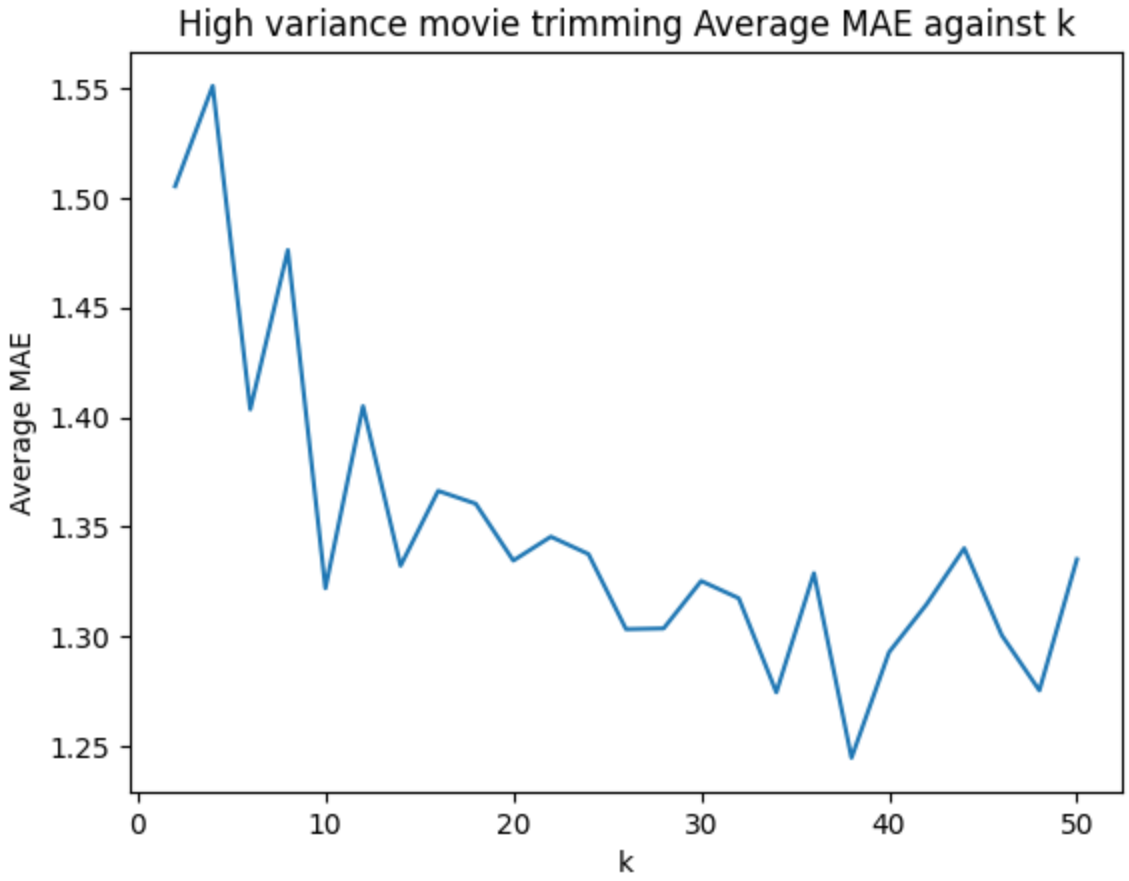


In [97]:
```
report_result_nmf(ratings, high_variance_trimming, "High variance movie trimming", ks=np
```

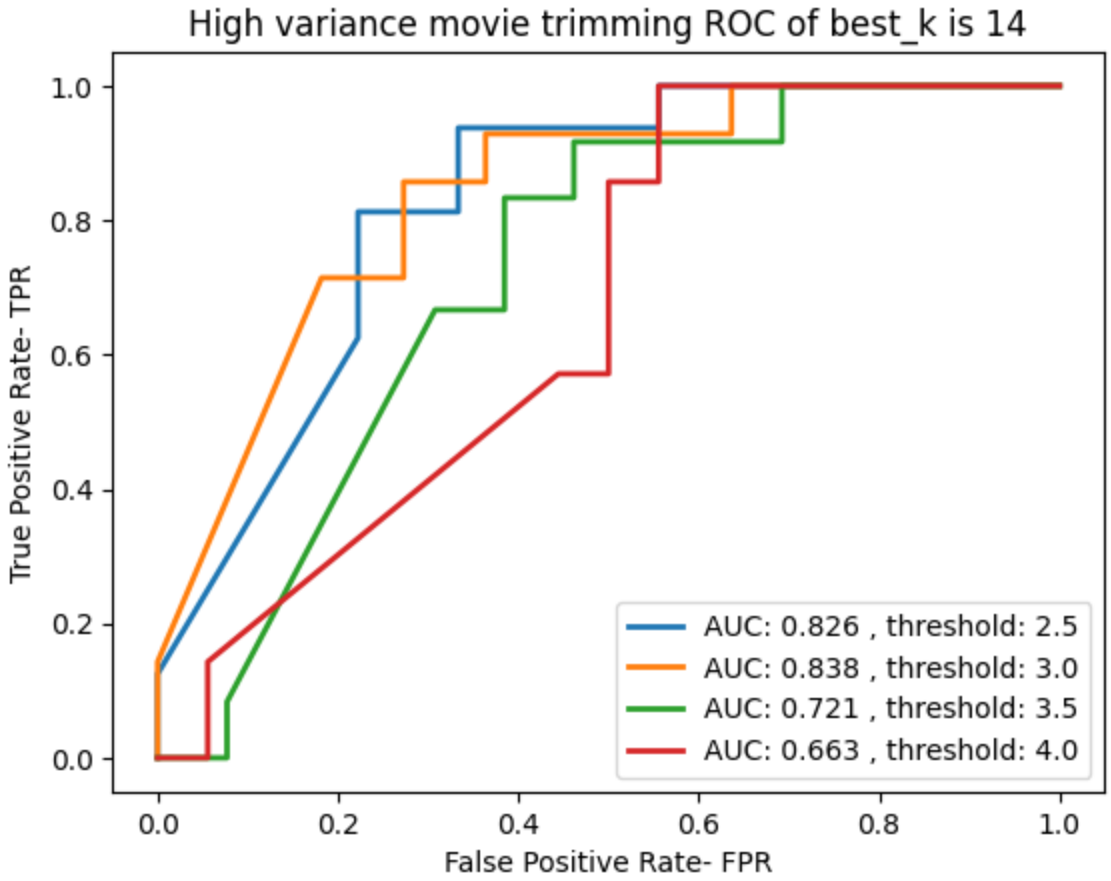
```
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```

k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=26:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=28:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=30:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=32:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=34:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=36:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=38:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=40:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=42:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=44:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=46:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=48:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=50:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,






```
High variance movie trimming Minimum k for RMSE is 14, average RMSE is 1.612
High variance movie trimming Minimum k for MAE is 26, average MAE is 1.303
```



Question 9

In [11]:
```
nmf_k20 = NMF(n_factors=20,n_epochs=30,verbose=False)
nmf_k20.fit(trainset).test(testset)
```

```
U_mat = nmf_k20.pu
V_mat = nmf_k20.qi

def interpret_latent_factor(num_latent_factors, V_mat):
    for i in range(num_latent_factors):
        movie_ids = np.argsort(-V_mat[:,i])[0:10]
        print(f"Latent Factor {i}")
        val = V_mat[movie_ids, i]
        for j in range(10):
          genre = movies.iloc[movie_ids[j]]['genres']
          print(f"Genre: {genre} Value: {val[j]:.4f}")
```

In [12]:
```
interpret_latent_factor(20, V_mat)
```

```
Latent Factor 0
Genre: Adventure|Drama Value: 2.1386
Genre: Comedy|Fantasy|Horror|Thriller Value: 1.9934
Genre: Action|Crime|Drama Value: 1.8720
Genre: Comedy|Crime|Drama Value: 1.8599
Genre: Comedy|Drama|Romance Value: 1.8268
Genre: Crime|Mystery Value: 1.7254
Genre: Horror|Thriller Value: 1.6945
Genre: Drama|Mystery|Thriller Value: 1.6634
Genre: Action|Thriller Value: 1.6603
Genre: Comedy|Romance Value: 1.6455
Latent Factor 1
Genre: Comedy Value: 2.1365
Genre: Action|Adventure|Animation|Fantasy|Sci-Fi Value: 2.0220
Genre: Comedy Value: 1.7544
Genre: Adventure|Animation|Children Value: 1.7514
Genre: Action|Animation|Children|Crime Value: 1.6294
Genre: Action|Adventure|Sci-Fi|Thriller Value: 1.6181
Genre: Comedy Value: 1.6069
Genre: Drama Value: 1.5759
Genre: Comedy Value: 1.5555
Genre: Drama Value: 1.5539
Latent Factor 2
Genre: Crime|Film-Noir Value: 2.0518
Genre: Drama|Thriller Value: 1.9304
Genre: Drama Value: 1.7626
Genre: Action|Crime|Drama Value: 1.7527
Genre: Action|Crime|Thriller Value: 1.6896
Genre: Horror|Thriller Value: 1.6653
Genre: Children|Comedy Value: 1.6411
Genre: Comedy|Romance Value: 1.6338
Genre: Drama|Film-Noir Value: 1.6238
Genre: Drama|Romance Value: 1.5968
Latent Factor 3
Genre: Drama|Romance Value: 2.4556
Genre: Comedy|Drama Value: 2.3891
Genre: Drama|Romance Value: 2.1252
Genre: Action|Comedy|IMAX Value: 1.9904
Genre: Drama Value: 1.9738
Genre: Drama Value: 1.9592
Genre: Action|Drama Value: 1.9540
Genre: Action|Crime|Drama|Mystery|Thriller Value: 1.8056
Genre: Children|Comedy|Romance Value: 1.7330
Genre: Drama|Mystery|Romance|Thriller Value: 1.6567
Latent Factor 4
Genre: Children|Drama Value: 2.5822
Genre: Comedy Value: 2.0549
Genre: Adventure|Children|Drama Value: 1.9600
Genre: Action|Adventure|Animation|Fantasy|Sci-Fi Value: 1.9284
Genre: Comedy Value: 1.9019
```

Genre: Children|Comedy Value: 1.8384
Genre: Drama|Romance Value: 1.8296
Genre: Action|Mystery|Sci-Fi|Thriller Value: 1.7452
Genre: Animation|Fantasy|Horror|Sci-Fi Value: 1.7419
Genre: Comedy|Drama|Romance Value: 1.7414
Latent Factor 5
Genre: Drama|Thriller Value: 2.6051
Genre: Action|Adventure|Sci-Fi Value: 2.4430
Genre: Horror|Thriller Value: 2.1767
Genre: Action|Crime|Drama Value: 1.8293
Genre: Comedy Value: 1.7877
Genre: Adventure|Comedy Value: 1.7806
Genre: Drama|Horror Value: 1.7644
Genre: Crime|Mystery Value: 1.7600
Genre: Drama Value: 1.7518
Genre: Comedy|Horror Value: 1.6550
Latent Factor 6
Genre: Comedy Value: 1.8187
Genre: Comedy|Western Value: 1.6718
Genre: Action|Adventure|Comedy|Fantasy Value: 1.5677
Genre: Horror|Sci-Fi|Thriller Value: 1.5616
Genre: Comedy|Drama Value: 1.5580
Genre: Drama Value: 1.5284
Genre: Children|Comedy|Drama Value: 1.4921
Genre: Horror|Mystery|Sci-Fi Value: 1.4859
Genre: Documentary Value: 1.4654
Genre: Comedy|Romance Value: 1.4595
Latent Factor 7
Genre: Comedy|Drama Value: 2.2399
Genre: Comedy|Drama|Romance Value: 2.0662
Genre: Crime|Drama|Fantasy|Film-Noir|Mystery|Romance Value: 1.8316
Genre: Crime|Drama|Thriller Value: 1.7977
Genre: Comedy|Fantasy|Romance Value: 1.7324
Genre: Comedy|Drama Value: 1.7093
Genre: Comedy|Drama|Romance Value: 1.7060
Genre: Comedy Value: 1.7025
Genre: Adventure|Children|Comedy|Mystery Value: 1.6586
Genre: Drama Value: 1.6545
Latent Factor 8
Genre: Comedy Value: 2.2946
Genre: Comedy Value: 2.0351
Genre: Drama|Thriller|War Value: 1.9875
Genre: Comedy|Sci-Fi Value: 1.9523
Genre: Drama Value: 1.9444
Genre: Comedy Value: 1.9270
Genre: Action|Comedy|Crime|Thriller Value: 1.8681
Genre: Comedy|Drama Value: 1.8361
Genre: Action|Thriller Value: 1.6528
Genre: Crime|Mystery|Thriller Value: 1.5259
Latent Factor 9
Genre: Documentary Value: 2.0359
Genre: Drama Value: 1.7556
Genre: Drama Value: 1.7229
Genre: Action|Crime Value: 1.6709
Genre: Comedy|Musical|Romance Value: 1.5868
Genre: Fantasy|Horror Value: 1.5861
Genre: Adventure|Fantasy|Sci-Fi Value: 1.5743
Genre: Action|Horror|Sci-Fi|Thriller Value: 1.5611
Genre: Comedy|Romance Value: 1.5609
Genre: Drama Value: 1.5580
Latent Factor 10
Genre: Comedy|Drama Value: 2.2501
Genre: Comedy Value: 1.8429
Genre: Adventure|Children|Fantasy|Sci-Fi Value: 1.7894
Genre: Drama Value: 1.7531
Genre: Adventure|Animation|Children|Drama|Fantasy Value: 1.7386

Genre: Drama|War Value: 1.7290
Genre: Action|Comedy|Drama Value: 1.7245
Genre: Drama Value: 1.6285
Genre: Drama|Sci-Fi|Thriller Value: 1.6241
Genre: Comedy|Romance Value: 1.5721
Latent Factor 11
Genre: Drama Value: 1.9061
Genre: Fantasy|Horror|Thriller Value: 1.8301
Genre: Children|Comedy Value: 1.7848
Genre: Comedy|Drama Value: 1.7680
Genre: Drama|Horror|Thriller Value: 1.7242
Genre: Drama|War Value: 1.7163
Genre: Crime|Drama|Fantasy|Film-Noir|Mystery|Romance Value: 1.6961
Genre: Drama Value: 1.6348
Genre: Comedy Value: 1.6264
Genre: Comedy Value: 1.6009
Latent Factor 12
Genre: Drama|Thriller Value: 2.5625
Genre: Drama Value: 2.3612
Genre: Adventure|Comedy|War Value: 2.2861
Genre: Adventure|Children|Drama Value: 2.0454
Genre: Comedy|Drama Value: 1.9340
Genre: Comedy Value: 1.9295
Genre: Drama|Romance Value: 1.9131
Genre: Comedy Value: 1.8981
Genre: Comedy|Crime Value: 1.8762
Genre: Drama|Romance Value: 1.8427
Latent Factor 13
Genre: Action|Mystery|Sci-Fi|Thriller Value: 2.1264
Genre: Comedy|Drama Value: 2.0597
Genre: Comedy|Romance Value: 2.0264
Genre: Comedy|Musical|Western Value: 1.9340
Genre: Drama Value: 1.9000
Genre: Drama Value: 1.8799
Genre: Children|Drama Value: 1.7453
Genre: Drama Value: 1.7422
Genre: Comedy Value: 1.7353
Genre: Drama|Horror|Thriller Value: 1.7137
Latent Factor 14
Genre: Drama Value: 2.1446
Genre: Action|Animation|Film-Noir|Sci-Fi|Thriller Value: 1.9276
Genre: Adventure|Children Value: 1.8539
Genre: Comedy|Romance Value: 1.8527
Genre: Comedy|Crime|Drama Value: 1.7733
Genre: Crime|Drama|Thriller Value: 1.7300
Genre: Adventure|Comedy|Drama|Fantasy Value: 1.6052
Genre: Comedy Value: 1.5662
Genre: Action|Adventure|Comedy|Crime|Drama|Film-Noir|Horror|Mystery|Thriller|Western Val
ue: 1.5316
Genre: Comedy|Crime Value: 1.5244
Latent Factor 15
Genre: Adventure|Drama|Sci-Fi Value: 2.7298
Genre: Action|Drama Value: 2.0109
Genre: Comedy Value: 1.9922
Genre: Documentary Value: 1.8175
Genre: Children|Comedy|Drama Value: 1.7733
Genre: Adventure|Fantasy|IMAX Value: 1.7550
Genre: Action|Crime|Drama|Thriller Value: 1.7241
Genre: Crime|Mystery Value: 1.7123
Genre: Documentary Value: 1.7098
Genre: Comedy|Romance Value: 1.7070
Latent Factor 16
Genre: Comedy|Crime Value: 1.9201
Genre: Horror|Sci-Fi|Thriller Value: 1.8048
Genre: Crime|Drama|Thriller Value: 1.7170
Genre: Documentary Value: 1.6994

```
Genre: Drama Value: 1.6940
Genre: Drama|Horror|Sci-Fi Value: 1.6840
Genre: Adventure|Western Value: 1.6764
Genre: Comedy|Horror Value: 1.6695
Genre: Action|Drama|Thriller Value: 1.6634
Genre: Comedy|Drama Value: 1.6441
Latent Factor 17
Genre: Comedy|Crime Value: 2.6101
Genre: Comedy Value: 1.7375
Genre: Comedy|Horror Value: 1.7210
Genre: Drama|Romance Value: 1.7036
Genre: Comedy|Romance Value: 1.6856
Genre: Action|Comedy|Sci-Fi Value: 1.6206
Genre: Action|Adventure|Drama|War Value: 1.6150
Genre: Comedy Value: 1.5919
Genre: Drama|Mystery|Sci-Fi Value: 1.5581
Genre: Action|Crime|Thriller Value: 1.5150
Latent Factor 18
Genre: Film-Noir|Mystery|Thriller Value: 2.3404
Genre: Action|Drama Value: 1.9885
Genre: Drama|Romance Value: 1.8636
Genre: Action|Adventure|Sci-Fi Value: 1.8625
Genre: Drama Value: 1.7317
Genre: Drama Value: 1.7181
Genre: Drama Value: 1.6399
Genre: Fantasy|Horror Value: 1.6232
Genre: Drama|Fantasy Value: 1.6058
Genre: Documentary|Horror Value: 1.5901
Latent Factor 19
Genre: Drama|Horror Value: 2.5958
Genre: Action|Comedy Value: 2.2388
Genre: Action|Crime|Fantasy|Sci-Fi|Thriller Value: 2.1026
Genre: Horror|Sci-Fi Value: 2.0842
Genre: Drama|Romance Value: 1.9748
Genre: Crime|Drama|Mystery|Thriller Value: 1.9715
Genre: Comedy|Fantasy|Horror|Thriller Value: 1.9579
Genre: Adventure|Animation|Children|Comedy|Fantasy|Romance Value: 1.9126
Genre: Crime|Drama|Thriller Value: 1.8599
Genre: Comedy|Horror Value: 1.8531
```

Question 10

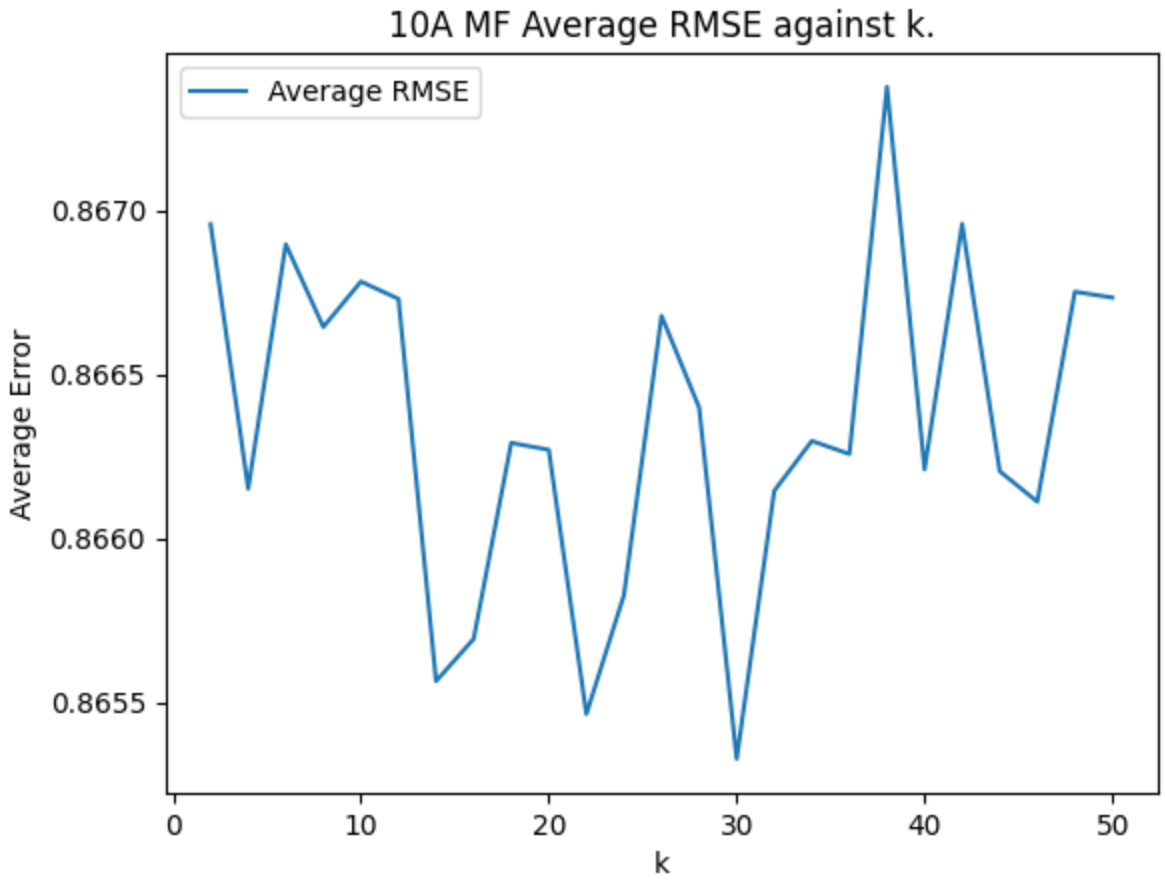
In [5]:

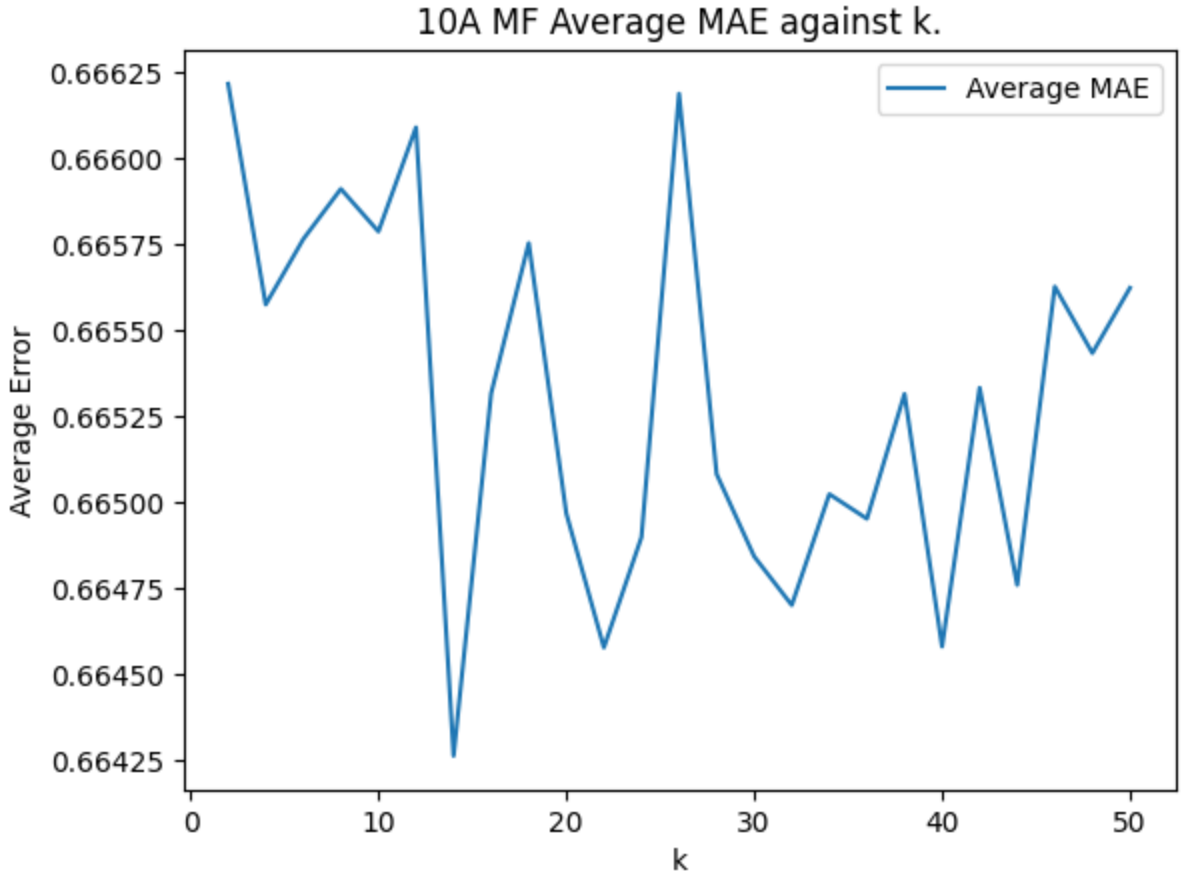
```
avg_rmse = []
avg_mae = []

min_rmse = 1e4
min_mae = 1e4
min_rmse_k = 0
min_rmsse_k = 0

new_ks = np.arange(2,52,2)
for i in trange(len(new_ks)):
  k = new_ks[i]
  print(k)
  svd = SVD(n_factors = k)
  cv_ = cross_validate(svd, data, measures=['rmse', 'mae'], cv=10, verbose=False)

  rmse = np.mean(cv_["test_rmse"])
  avg_rmse.append(rmse)
  if rmse < min_rmse:
    min_rmse = rmse
    min_rmse_k = k

  mae = np.mean(cv_["test_mae"])
```

```
  avg_mae.append(mae)
  if mae < min_mae:
    min_mae = mae
    min_mae_k = k

plt.plot(new_ks, avg_rmse, label="Average RMSE")
plt.title("10A MF Average RMSE against k.")
plt.legend()
plt.xlabel("k")
plt.ylabel("Average Error")
plt.show()

plt.plot(new_ks, avg_mae, label="Average MAE")
plt.title("10A MF Average MAE against k.")
plt.legend()
plt.xlabel("k")
plt.ylabel("Average Error")
plt.show()
```

```
  0%|          | 0/25 [00:00<?, ?it/s]
2
  4%|▍         | 1/25 [00:06<02:40,  6.69s/it]
4
  8%|▊         | 2/25 [00:14<02:45,  7.19s/it]
6
 12%|█▏        | 3/25 [00:22<02:48,  7.65s/it]
8
 16%|█▌        | 4/25 [00:30<02:45,  7.88s/it]
10
 20%|██        | 5/25 [00:39<02:41,  8.07s/it]
12
 24%|██▍       | 6/25 [00:47<02:36,  8.22s/it]
14
 28%|██▊       | 7/25 [00:56<02:30,  8.36s/it]
16
 32%|███▏      | 8/25 [01:04<02:23,  8.46s/it]
18
 36%|███▌      | 9/25 [01:13<02:17,  8.60s/it]
20
 40%|████      | 10/25 [01:23<02:11,  8.79s/it]
22
 44%|████▍     | 11/25 [01:37<02:28, 10.62s/it]
24
 48%|████▊     | 12/25 [01:55<02:44, 12.66s/it]
26
 52%|█████▏    | 13/25 [02:12<02:48, 14.08s/it]
28
 56%|█████▌    | 14/25 [02:21<02:16, 12.45s/it]
30
 60%|██████    | 15/25 [02:28<01:50, 11.06s/it]
32
 64%|██████▍   | 16/25 [02:40<01:40, 11.18s/it]
34
 68%|██████▊   | 17/25 [02:49<01:24, 10.55s/it]
36
 72%|███████▏  | 18/25 [02:58<01:10, 10.13s/it]
38
 76%|███████▌  | 19/25 [03:06<00:56,  9.48s/it]
40
```

```
 80%|████████  | 20/25 [03:15<00:46,  9.22s/it]
42
 84%|████████▍ | 21/25 [03:23<00:35,  8.94s/it]
44
 88%|████████▊ | 22/25 [03:32<00:26,  8.88s/it]
46
 92%|█████████▏| 23/25 [03:41<00:18,  9.12s/it]
48
 96%|█████████▌| 24/25 [03:51<00:09,  9.19s/it]
50
100%|██████████| 25/25 [04:03<00:00,  9.74s/it]
```





In [8]:
```
print("Question10B")
print(f"Minimum Average RMSE (SVD) = {min_rmse:.3f}, k = {min_rmse_k}")
print(f"Minimum Average MAE (SVD) = {min_mae:.3f}, k = {min_mae_k}")
```

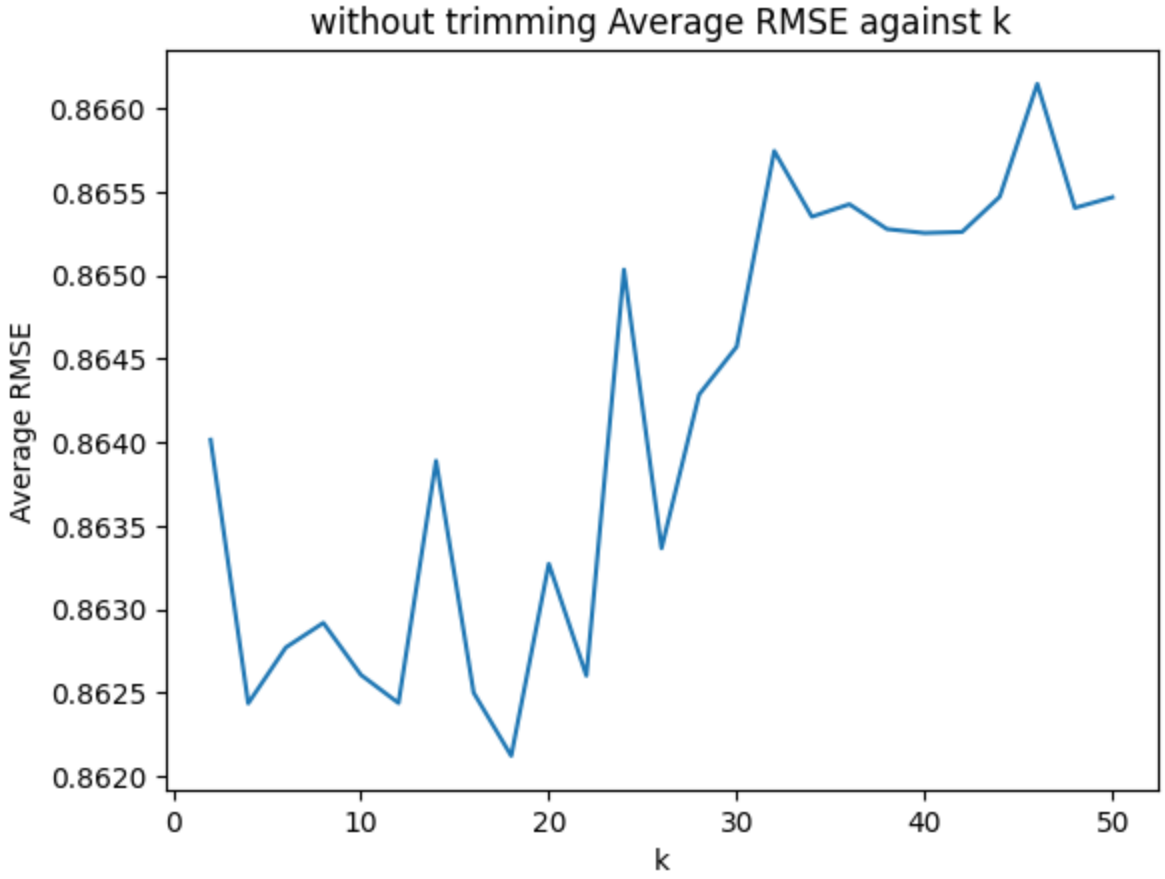
```
Question10B
  Minimum Average RMSE (SVD) = 0.865, k = 24
  Minimum Average MAE (SVD) = 0.664, k = 24
```

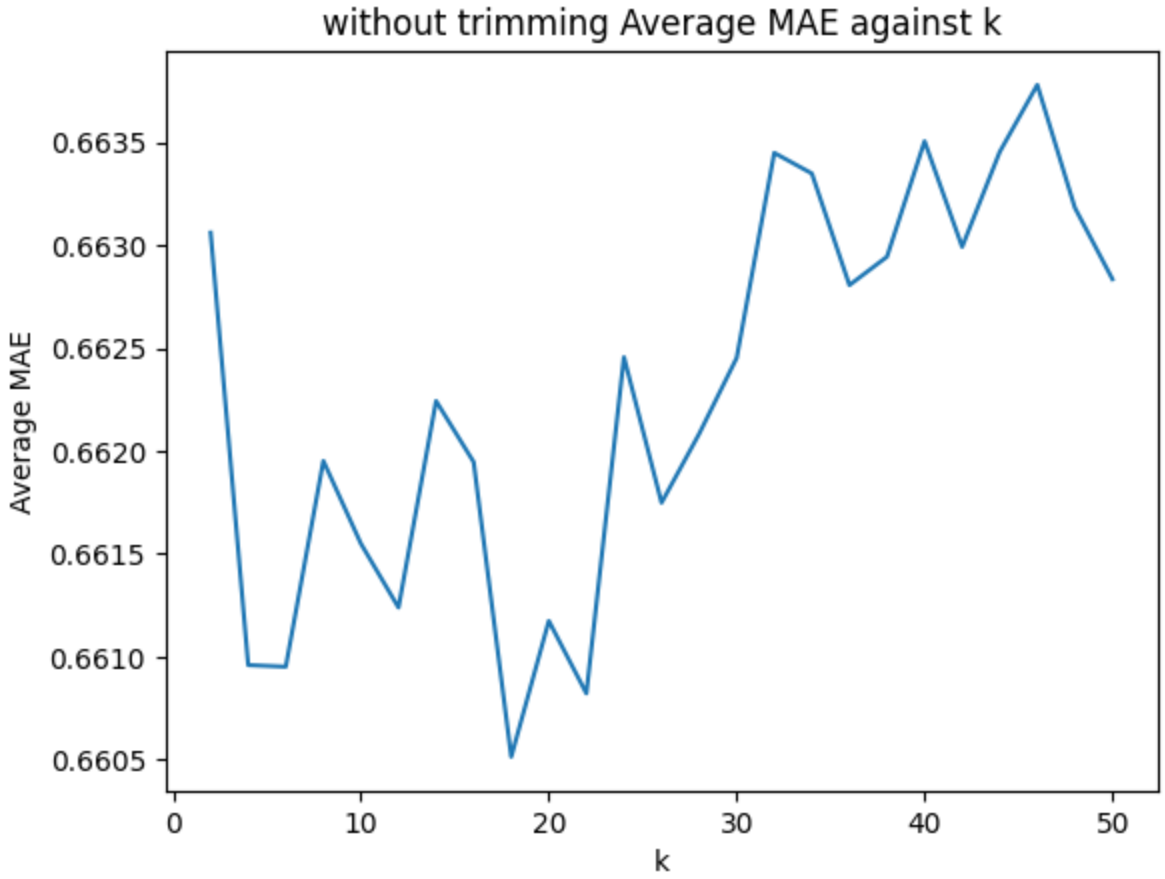
In [90]:
```
def report_roc_svd(trainset, testset, best_k, title):
  Threshold_list = [2.5, 3.0, 3.5, 4.0]
  res  = SVD(n_factors=best_k,n_epochs=30,verbose=False).fit(trainset).test(testset)

  for thre in Threshold_list:
    thresholded_out = []
    for row in res:
        if row.r_ui > thre:
            thresholded_out.append(1)
        else:
            thresholded_out.append(0)
    FPR, TPR, thresholds = roc_curve(thresholded_out, [row.est for row in res])
    labels = f"AUC: {auc(FPR,TPR):.3f} , threshold: {thre}"
    plt.plot(FPR, TPR,lw=2, label=labels)

  plt.legend(loc='best')
  plt.title(f'{title} ROC of best_k is {best_k}')
  plt.ylabel('True Positive Rate- TPR')
  plt.xlabel('False Positive Rate- FPR')
  plt.show()

def report_result_svd(ratings, filter, title, ks):
  filter_ratings = ratings[ratings.apply(filter, axis=1)]
  data = Dataset.load_from_df(filter_ratings[['userId','movieId','rating']], reader=read

  avg_rmse = []
  avg_mae = []
  rmse_min = 1e3
```

```
  rmse_min_k = ks[-1]
  mae_min = 1e3
  mae_min_k = ks[-1]

  for k in ks:
      print(f"k={k}", end=":")
      rmse = 0
      mae = 0
      iter = 1
      for trainset, testset in kf.split(data):
          print(f"iter={iter}", end=",")
          iter += 1
          perf = SVD(n_factors=k,n_epochs=30,verbose=False).fit(trainset).test(testset)
          rmse += accuracy.rmse(perf,verbose=False)
          mae += accuracy.mae(perf,verbose=False)
      print("")
      if rmse < rmse_min:
        rmse_min = rmse
        rmse_min_k = k
      if mae < mae_min:
        mae_min = mae
        mae_min_k = k
      avg_rmse.append(rmse / 10.0)
      avg_mae.append(mae / 10.0)

  plot_k(title, avg_rmse, avg_mae,ks)
  best_k = minimum_k(title, avg_rmse, avg_mae,ks)
  if best_k == -1:
    print("no convergence, we just choose a k with min rmse")
    best_k = rmse_min_k
  report_roc_nmf(trainset, testset, best_k,title)
```

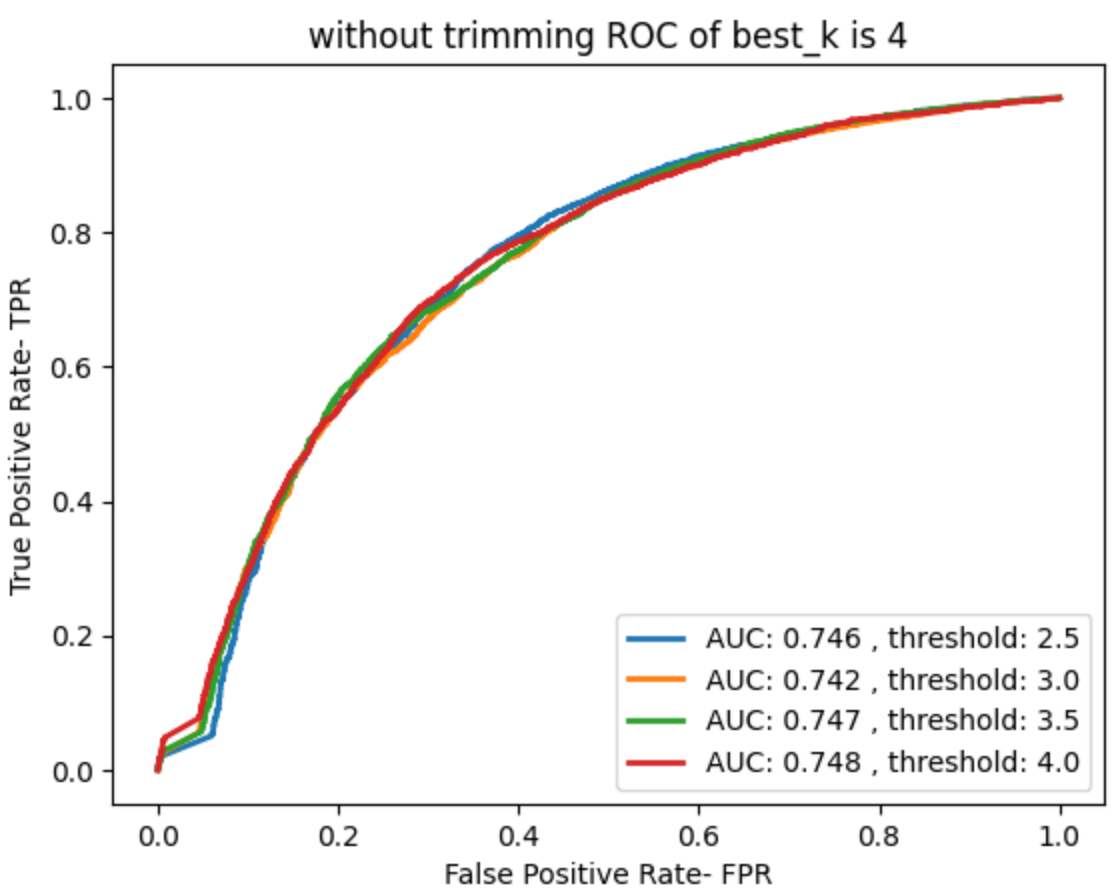
In [91]:
```
report_result_svd(ratings, do_nothing, "without trimming", ks=np.arange(2,52,2))
```

```
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=26:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=28:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=30:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=32:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=34:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=36:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=38:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=40:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=42:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=44:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=46:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=48:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=50:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```



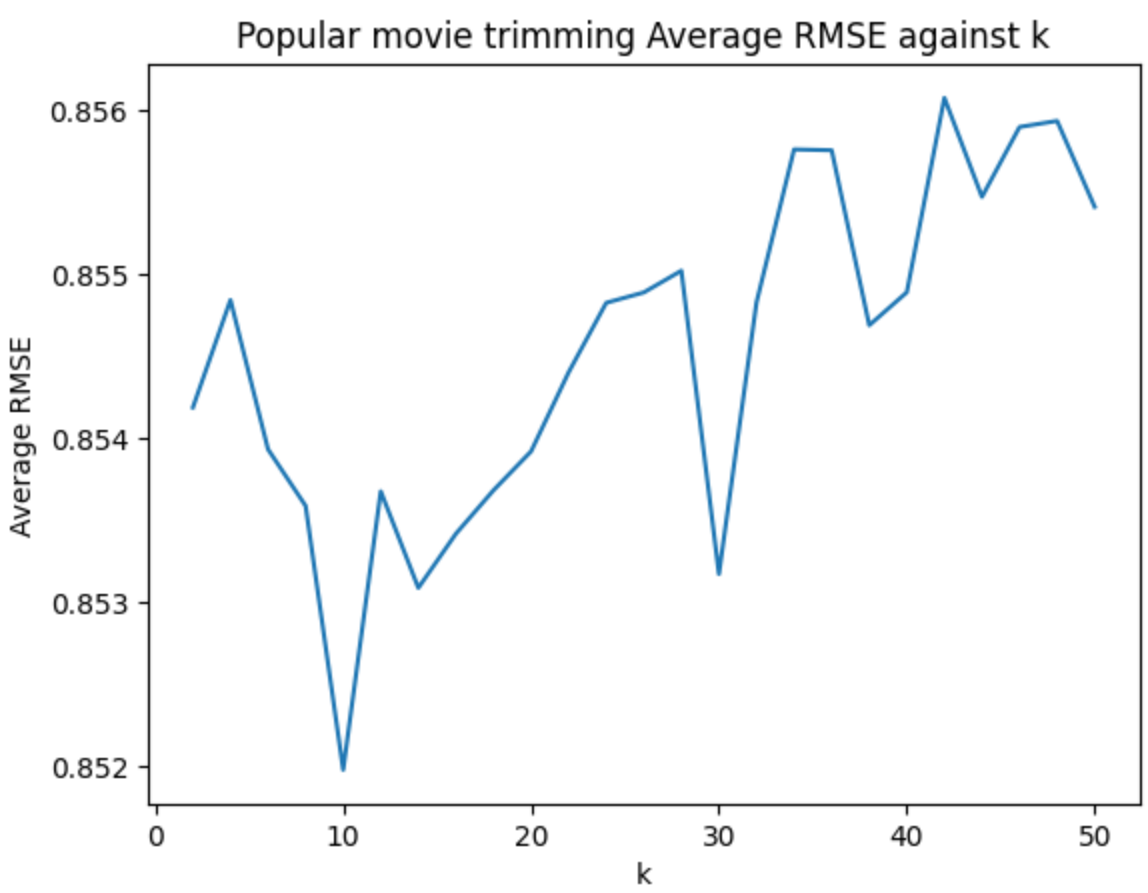


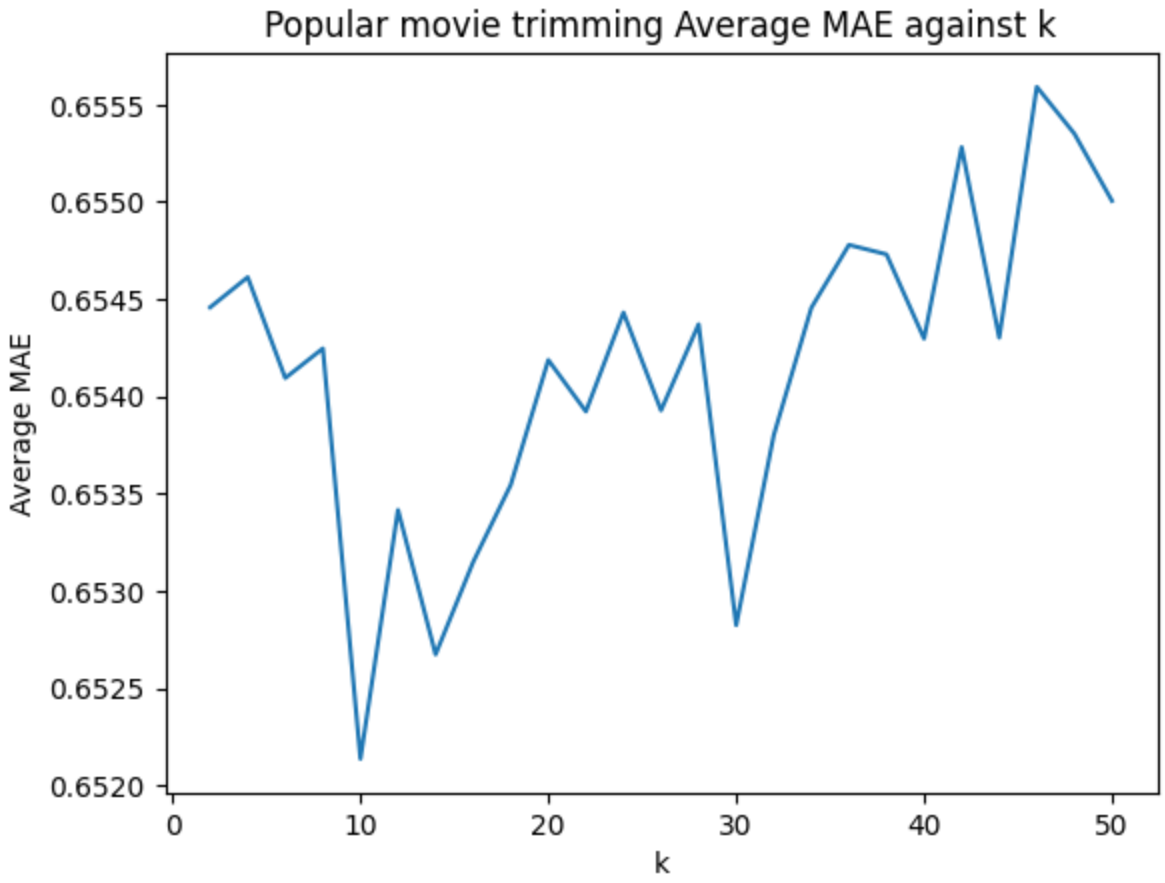

```
without trimming Minimum k for RMSE is 4, average RMSE is 0.862
without trimming Minimum k for MAE is 4, average MAE is 0.661
```



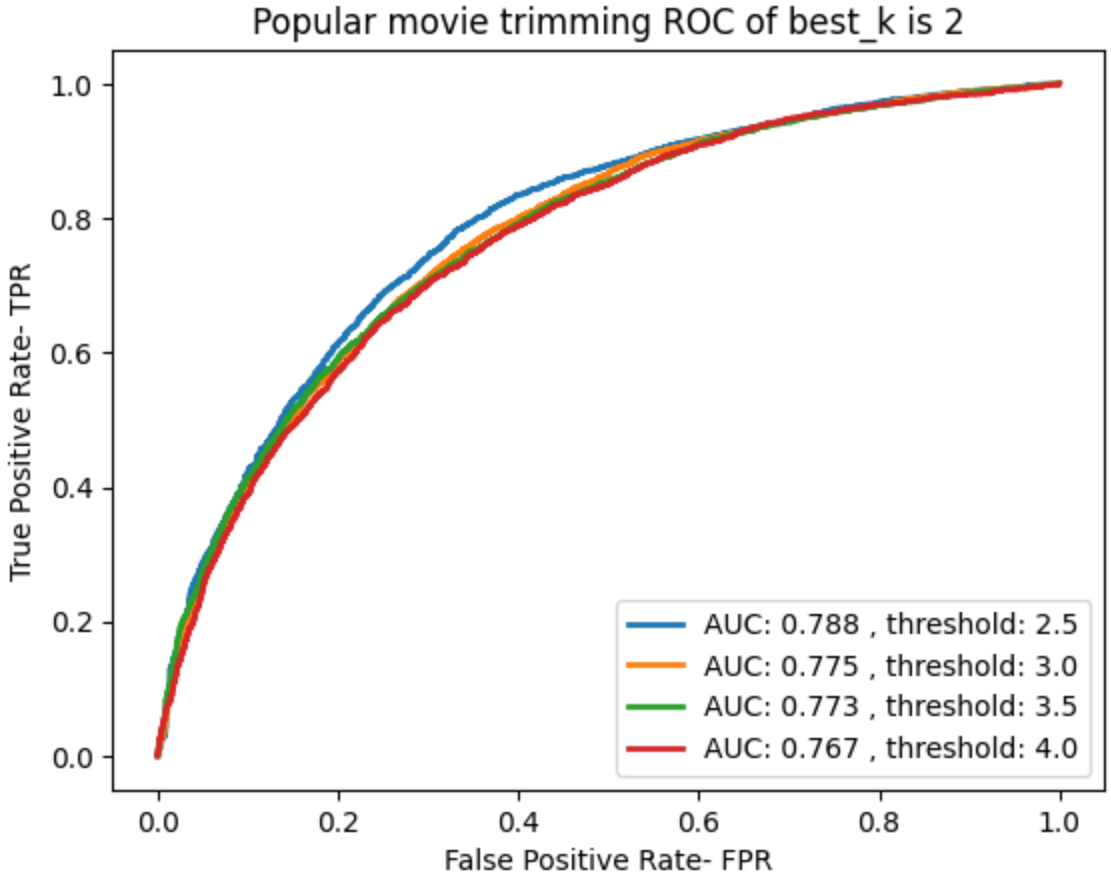
In [92]:
```
report_result_svd(ratings, popular_trimming, "Popular movie trimming", ks=np.arange(2,52
```

```
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=26:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=28:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=30:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=32:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=34:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=36:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=38:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=40:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=42:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=44:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=46:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=48:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=50:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```



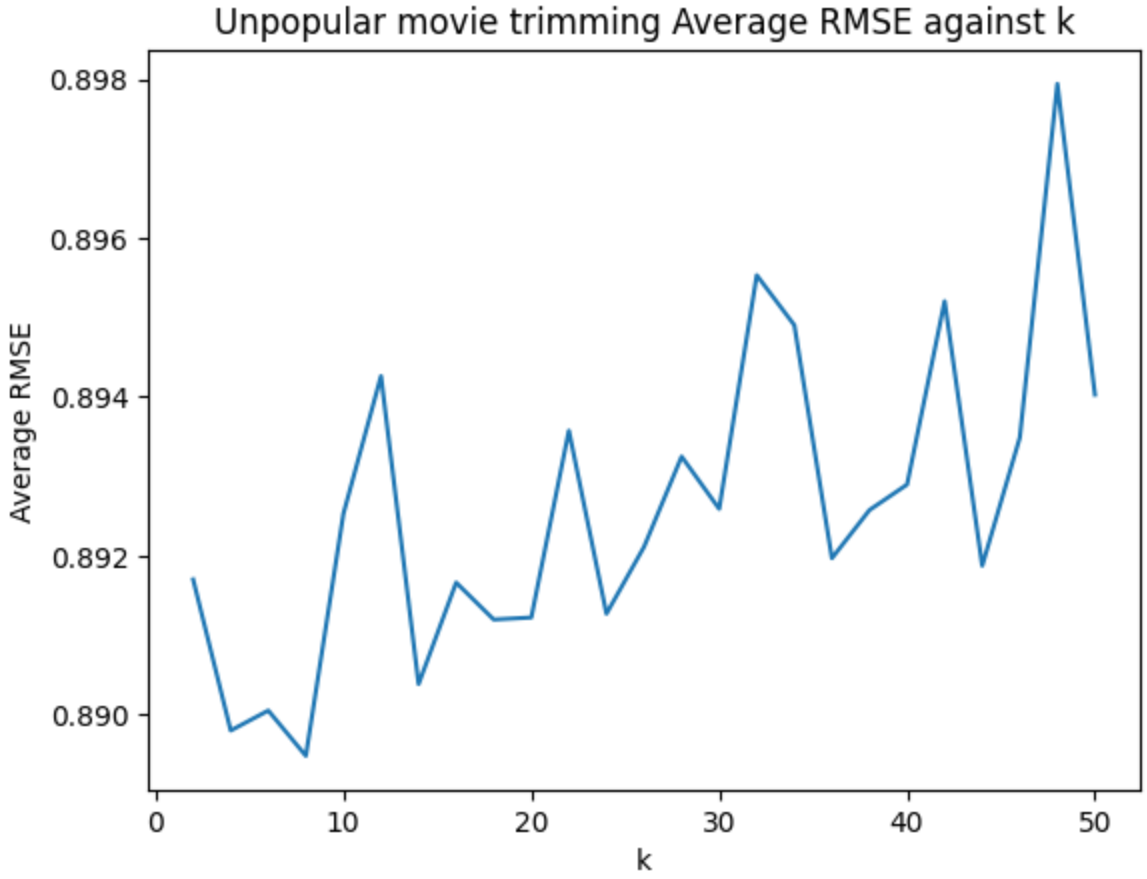


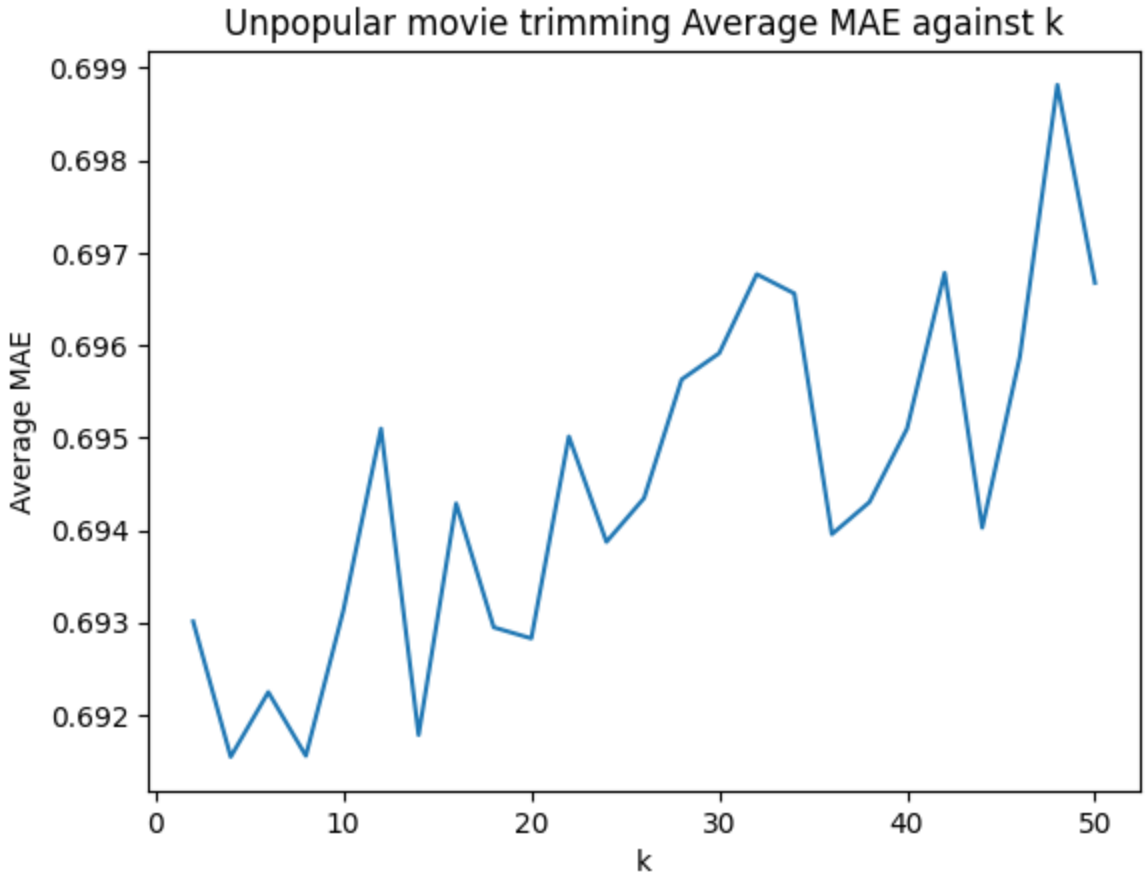

```
Popular movie trimming Minimum k for RMSE is 2, average RMSE is 0.854
Popular movie trimming Minimum k for MAE is 2, average MAE is 0.654
```



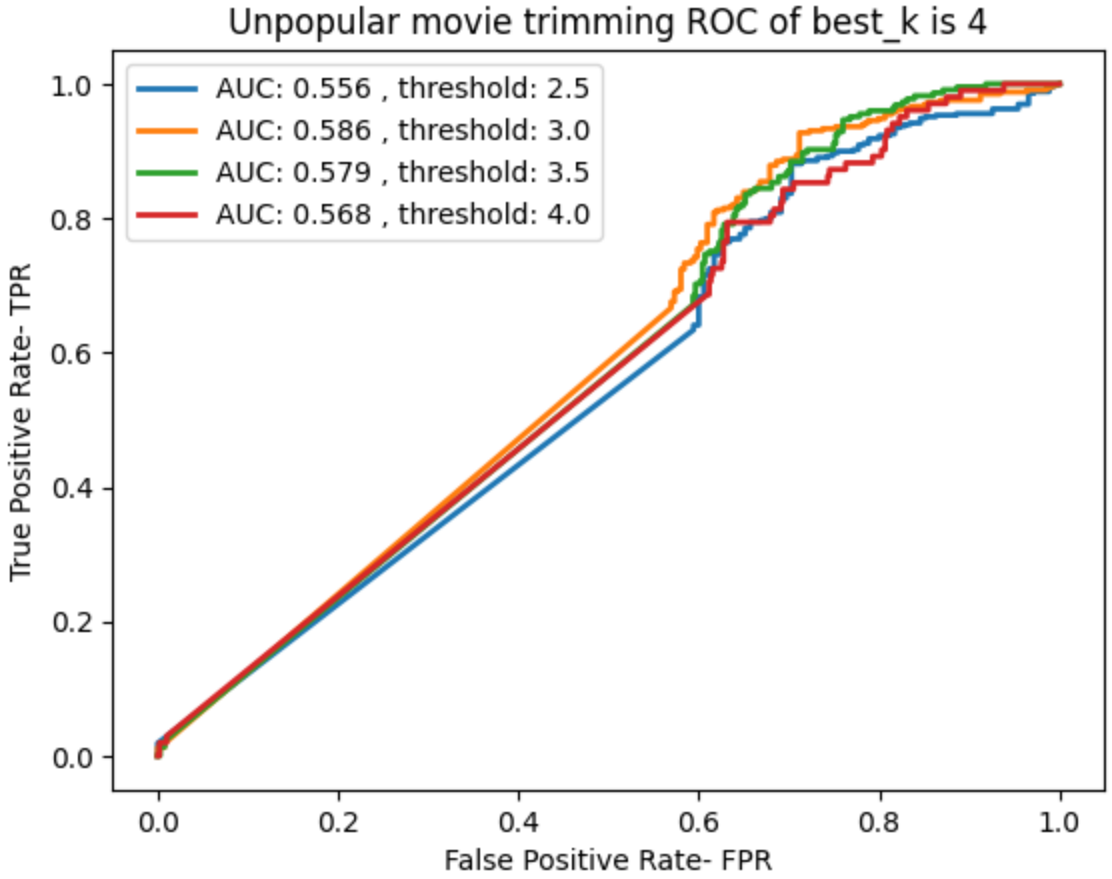
In [93]:
```
report_result_svd(ratings, unpopular_trimming, "Unpopular movie trimming", ks=np.arange(
```

```
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=26:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=28:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=30:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=32:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=34:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=36:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=38:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=40:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=42:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=44:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=46:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=48:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=50:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```



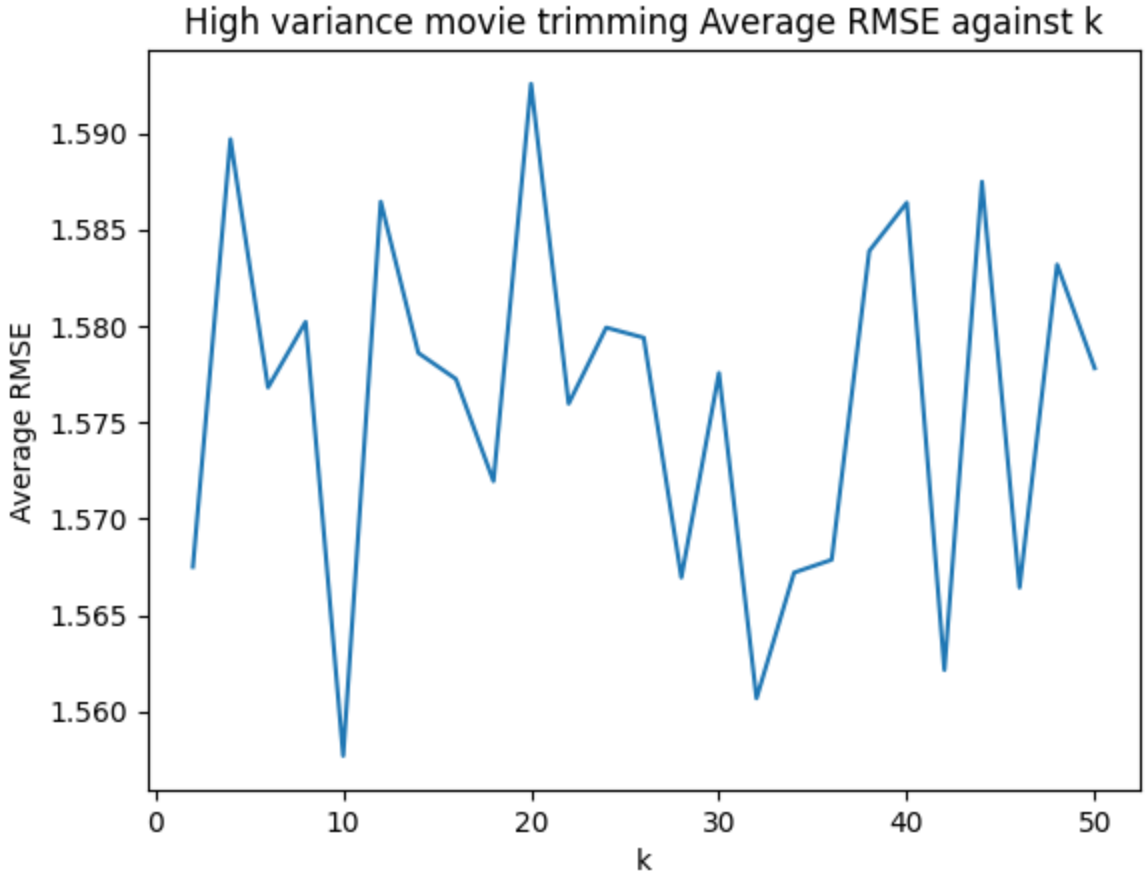


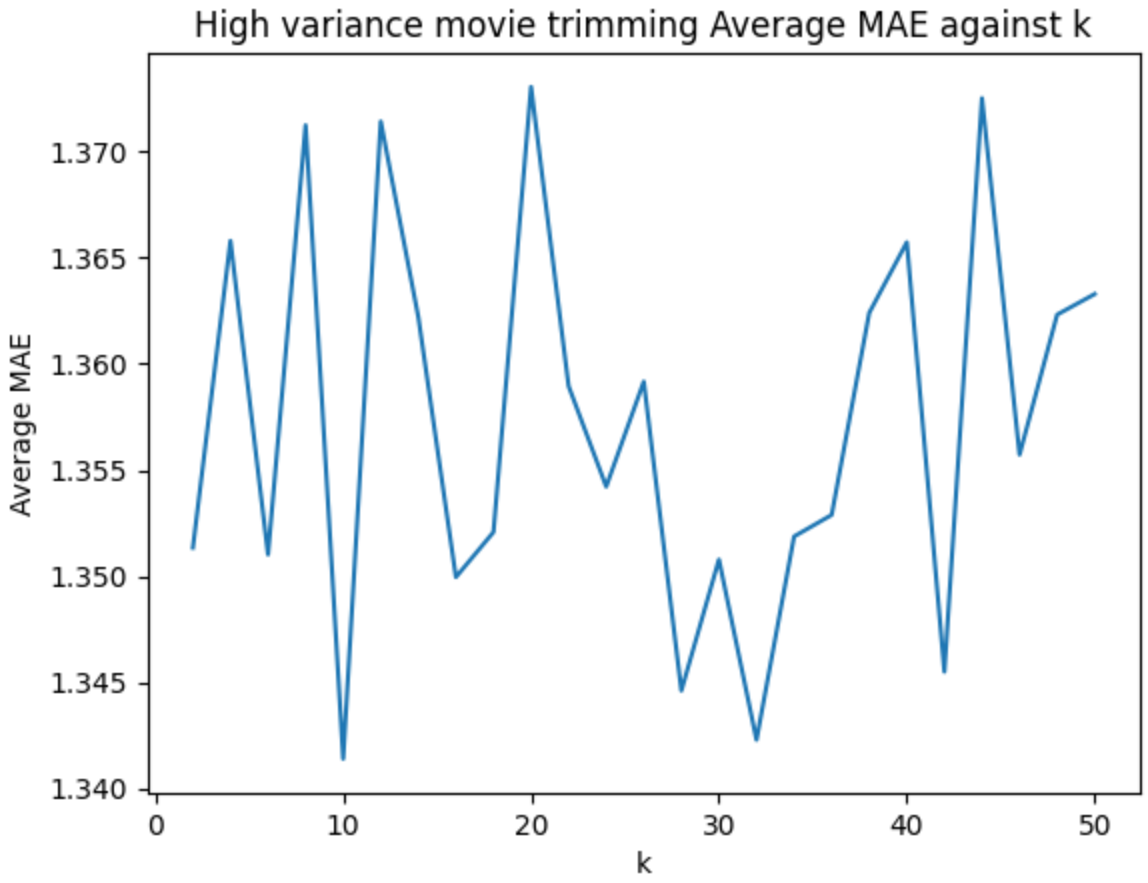

```
Unpopular movie trimming Minimum k for RMSE is 4, average RMSE is 0.890
Unpopular movie trimming Minimum k for MAE is 4, average MAE is 0.692
```



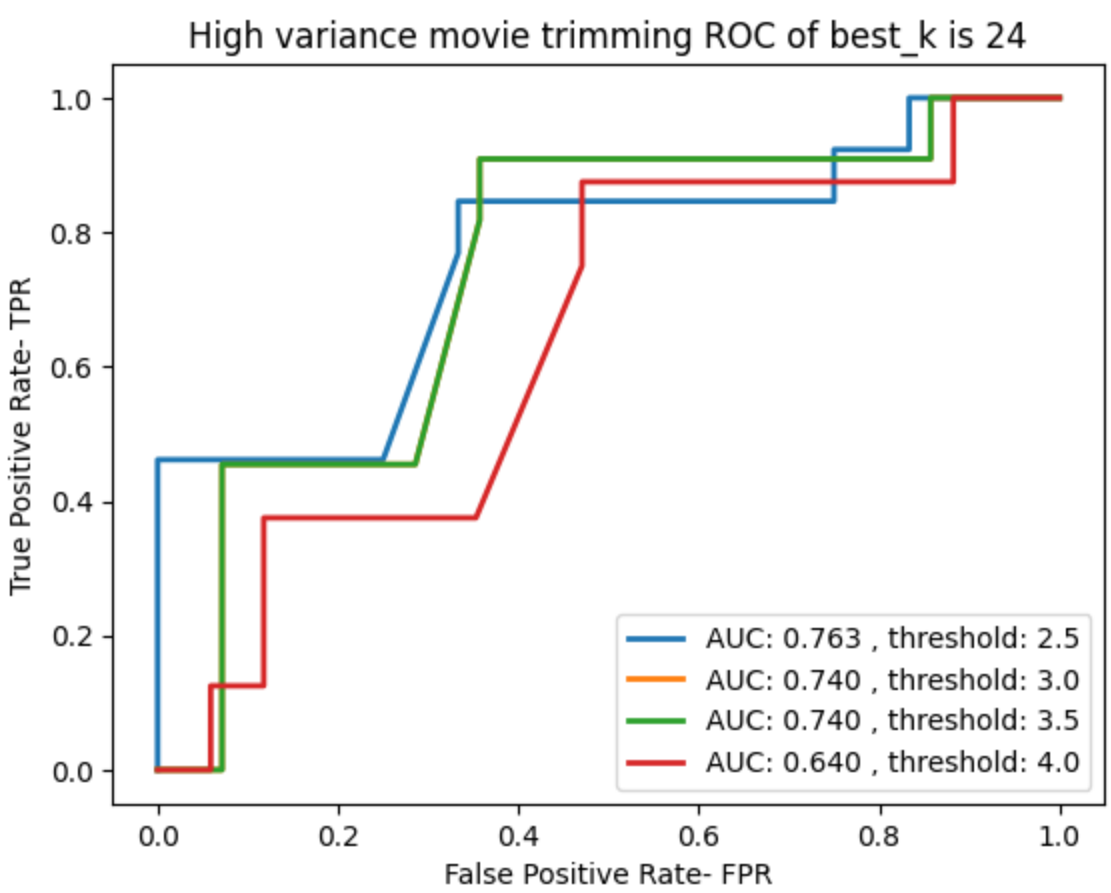
In [104…
```
report_result_svd(ratings, high_variance_trimming, "High variance movie trimming", ks=np
```

```
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=26:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=28:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=30:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=32:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=34:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=36:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=38:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=40:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=42:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=44:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=46:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=48:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=50:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```






```
High variance movie trimming Minimum k for RMSE is 24, average RMSE is 1.580
High variance movie trimming Minimum k for MAE is 48, average MAE is 1.362
```



Question 11

In [22]:
```
users_ratings = defaultdict(list)
for d in data.raw_ratings:
  user_id, movie_id, rating, _ = d
  users_ratings[user_id].append(rating)

users_mean_ratings = defaultdict(list)
for user_id in users_ratings.keys():
  users_mean_ratings[user_id] = np.mean(users_ratings[user_id])

kf = KFold(n_splits=10)
rmse_naive = 0

for trainset, testset in kf.split(data):
    pred = [users_mean_ratings[i[0]] for i in testset]
    true = [i[2] for i in testset]
    rmse_naive += np.sqrt(mean_squared_error(true, pred))

avg_rmse_naive = rmse_naive / 10.0

print(f"avg_rmse_naive = {avg_rmse_naive:.3f}")
```

```
avg_rmse_naive = 0.935
```

In [25]:
```
def report_result_naive(function,title):
    rmse_naive = 0
    for trainset, testset in kf.split(data):
        trim = function(testset)
        pred = [users_mean_ratings[i[0]] for i in trim]
        true = [i[2] for i in trim]
        rmse_naive += np.sqrt(mean_squared_error(true, pred))
    avg_rmse_naive = rmse_naive / 10.0
    print(f"naive collaborative filter {title} avg_rmse_naive = {avg_rmse_naive:.3f}")
```

In [29]:
```
report_result_naive(popular_trimming, "Popular movie trimming")
```

```
naive collaborative filter Popular movie trimming avg_rmse_naive = 0.932
```

In [28]:
```
report_result_naive(unpopular_trimming, "Unpopular movie trimming")
```

```
naive collaborative filter Unpopular movie trimming avg_rmse_naive = 0.971
```

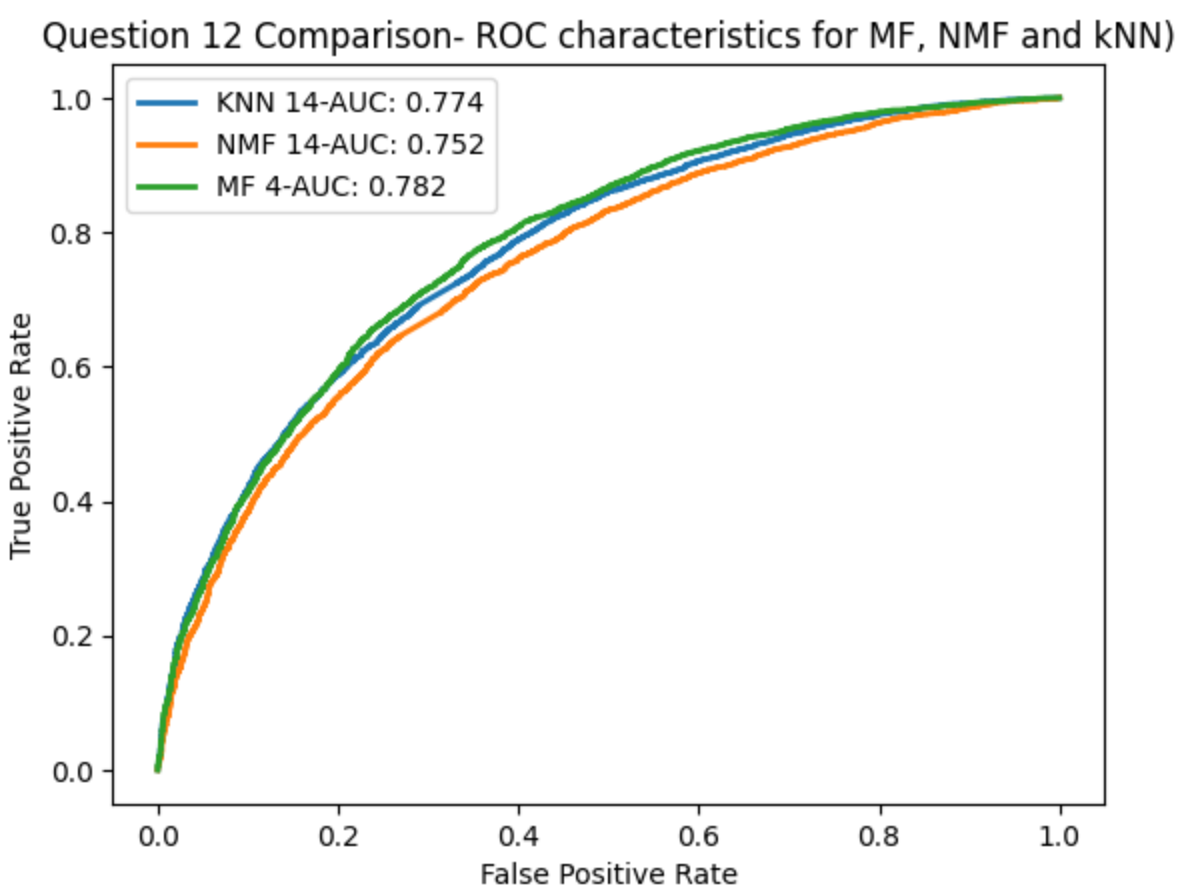
In [27]:
```
report_result_naive(high_variance_trimming, "High variance movie trimming")
```

```
naive collaborative filter High variance movie trimming avg_rmse_naive = 1.455
```

Question 12

In [ ]:
```
# KNN BEST_K = 14
KNNMeans_best = KNNWithMeans(k=14,sim_options={'name':'pearson'},verbose=False).fit(trai
# NMF BEST_K = 14
NMF_best  = NMF(n_factors=14,n_epochs=30,verbose=False).fit(trainset).test(testset)
# MF BEST_K = 4
MF_best  = SVD(n_factors=4,n_epochs=30,verbose=False).fit(trainset).test(testset)
```

In [9]:
```
def plot_(result,title):
    threshold_result = []
    for row in result:
        if row.r_ui > 3:
            threshold_result.append(1)
        else:
            threshold_result.append(0)
    fpr, tpr, thresholds = roc_curve(threshold_result, [row.est for row in result])
    labels = f"{title}-AUC: {auc(fpr,tpr):.3f}"
    plt.plot(fpr, tpr,lw=2, label=labels)

plot_(KNNMeans_best, "KNN 14")
plot_(NMF_best, "NMF 14")
plot_(MF_best, "MF 4")

plt.legend(loc='best')
plt.title('Question 12 Comparison- ROC characteristics for MF, NMF and kNN)')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
```



Question 13 is stated on our report

Question 14

In [108…

```
K = np.arange(1,26,1)
k_fold = KFold(n_splits=10)

dict_above_3 = defaultdict(list)
dict_of_all = defaultdict(list)

for _, row in ratings.iterrows():
    if row['rating'] >= 3.0:
        dict_above_3[row['userId']].append(row['movieId'])
    dict_of_all[row['userId']].append(row['movieId'])

def filter_(row,k):
  return len(dict_of_all[row['userId']]) >= k and len(dict_above_3[row['userId']]) > 0

def plot_precision_recall(function,pre,rec):
    plt.plot(K,pre)
    plt.title(f'{function}- Precision versus t')
    plt.ylabel(f'Average Precision-{function}')
    plt.xlabel('t')
    plt.show()

    plt.plot(K,rec)
    plt.title(f'{function}- Recall versus t')
    plt.ylabel(f'Average Recall-{function}')
    plt.xlabel('t')
    plt.show()

    plt.plot(rec, pre)
    plt.title(f'{function}- Precision versus Recall')
    plt.ylabel('Average precision')
```

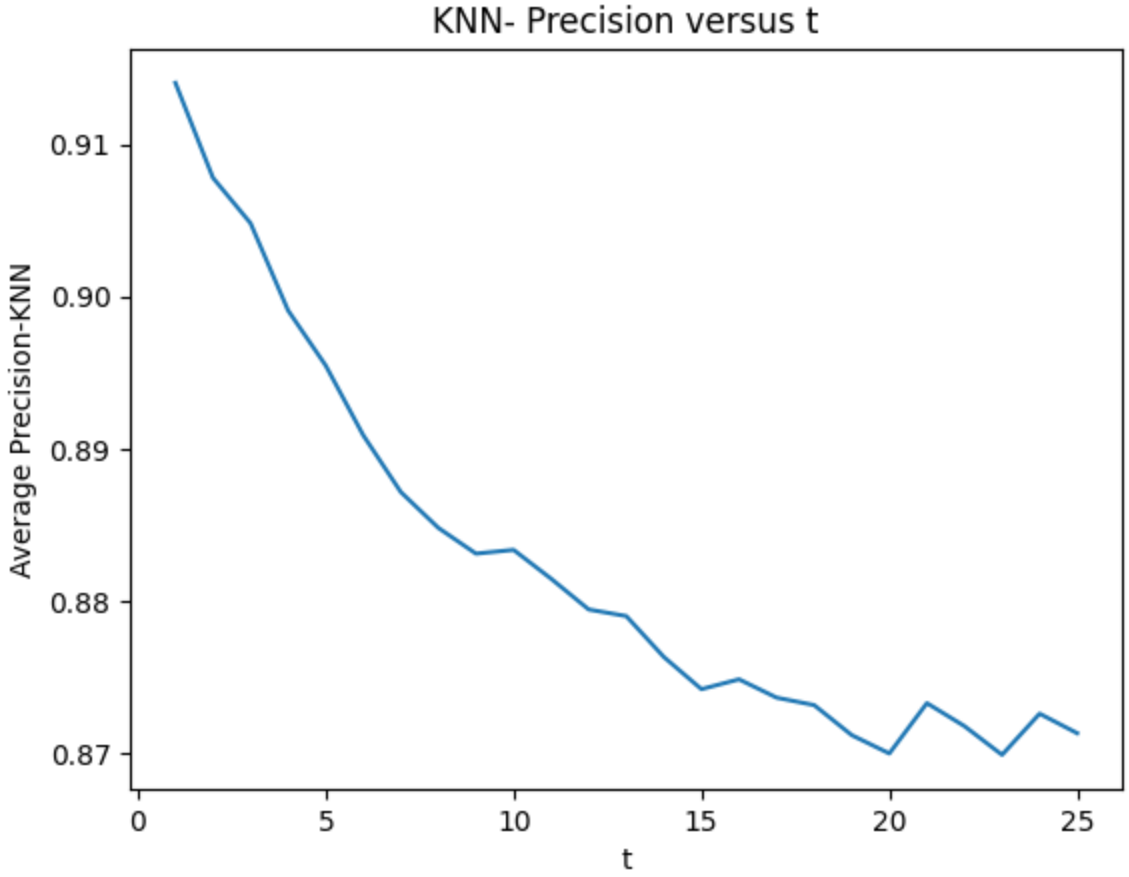
```
    plt.xlabel('Average recall')
    plt.show()

def calculate_precision_recall(ratings, function):
    precision_ = []
    recall_ = []

    for k in K:
        print(f"k={k}", end=":")
        iter = 1
        precision_k = []
        recall_k = []
        fuc_ = lambda x: filter_(x, k)
        filter_ratings = ratings[ratings.apply(fuc_, axis=1)]
        data = Dataset.load_from_df(filter_ratings[['userId','movieId','rating']], reade
        for trainset, testset in kf.split(data):
            print(f"iter={iter}", end=",")
            iter += 1
            if function == "KNN":
                res = KNNWithMeans(k=14,sim_options={'name':'pearson'},verbose=False).fi
            if function == "NMF":
                res = NMF(n_factors=14,n_epochs=30,verbose=False).fit(trainset).test(tes
            if function == "MF":
                res = SVD(n_factors=4,n_epochs=30,verbose=False).fit(trainset).test(test

            pred_ratings = defaultdict(list)

            for row in res:
                pred_ratings[row[0]].append((row[1],row[3]))

            precision_u = []
            recall_u = []
            for key in pred_ratings.keys():

                Set_all = sorted(pred_ratings[key],key=lambda x:x[1],reverse=True)
                Set_K = set([row[0] for row in Set_all[0:k]])

                precision_u.append(len(Set_K.intersection(dict_above_3[key]))/float(len(
                recall_u.append(len(Set_K.intersection(dict_above_3[key]))/float(len(dic

            precision_k.append(np.mean(precision_u))
            recall_k.append(np.mean(recall_u))
        print("")
        precision_.append(np.mean(precision_k))
        recall_.append(np.mean(recall_k))
    plot_precision_recall(function, precision_, recall_)
    return precision_, recall_
```

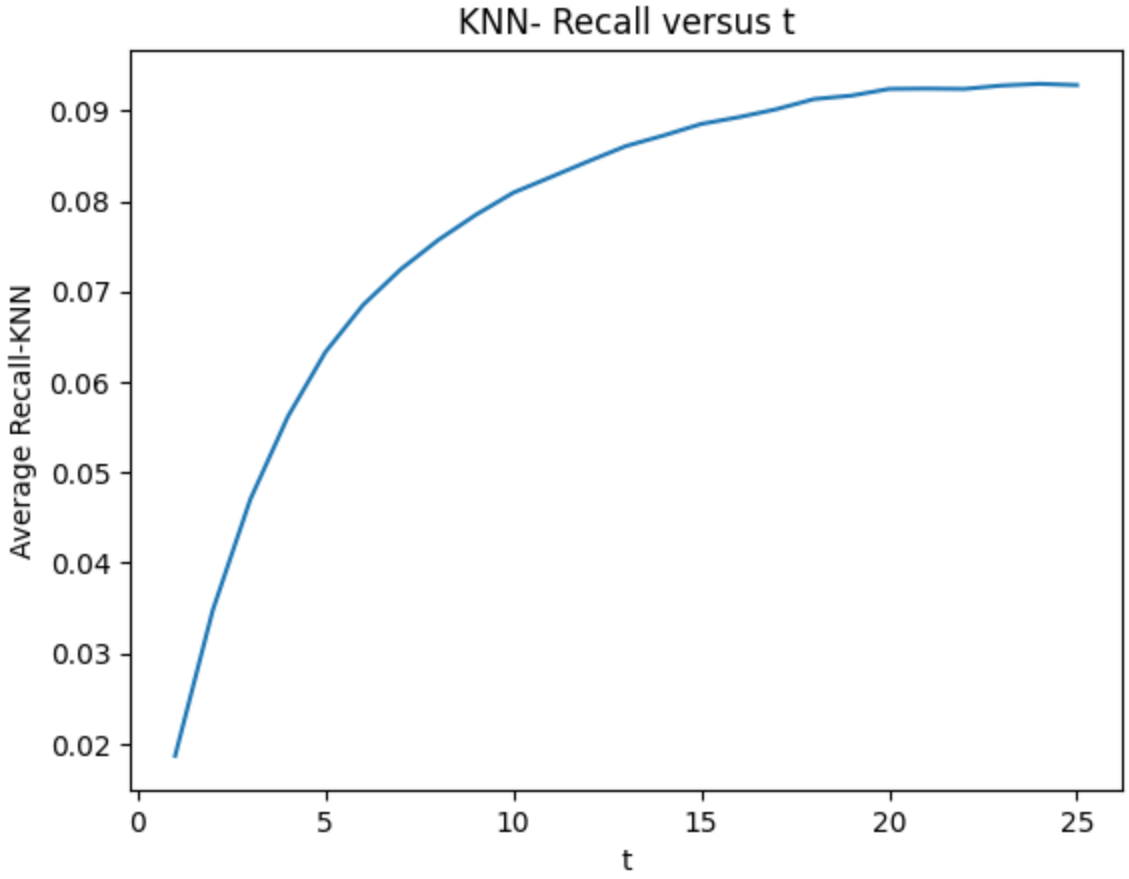
In [109…

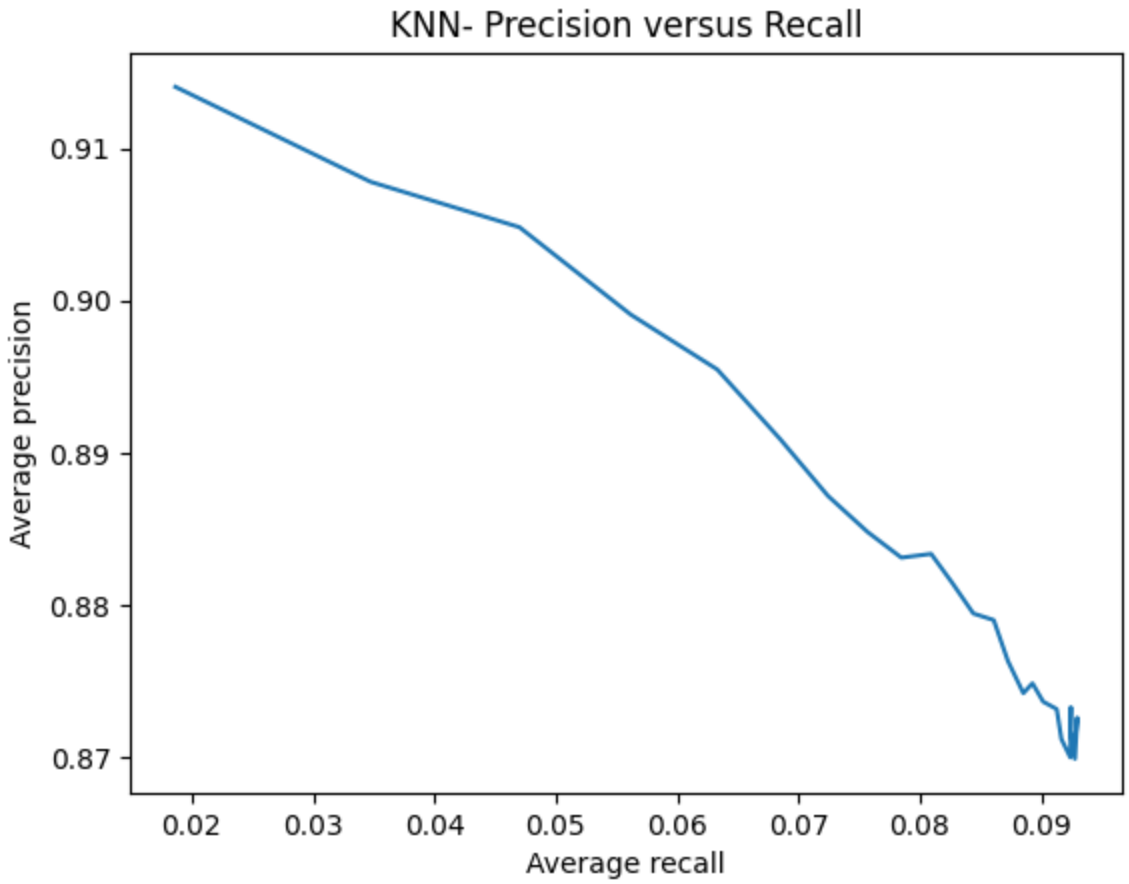
```
pre_knn, rec_knn = calculate_precision_recall(ratings, "KNN")
```

```
k=1:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=3:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=5:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=7:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=9:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=11:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=13:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=15:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```

k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=17:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=19:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=21:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=23:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=25:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,





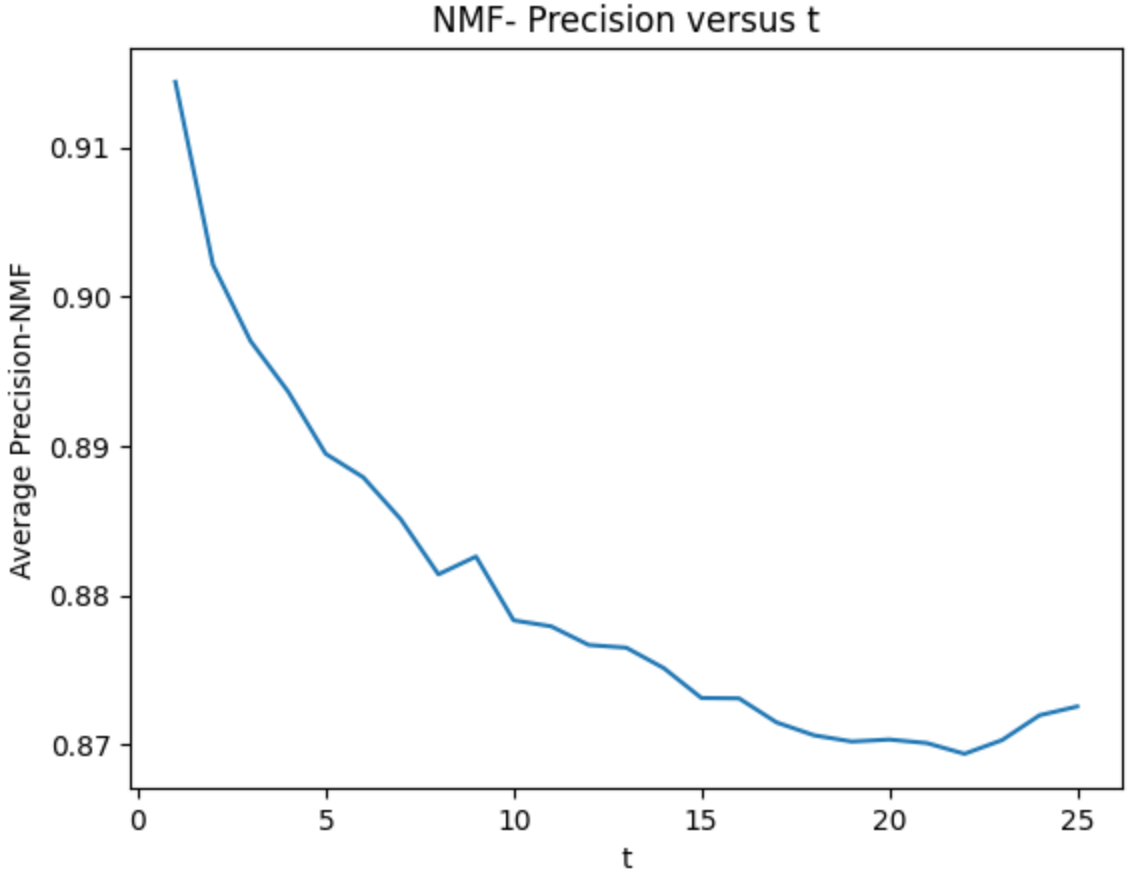


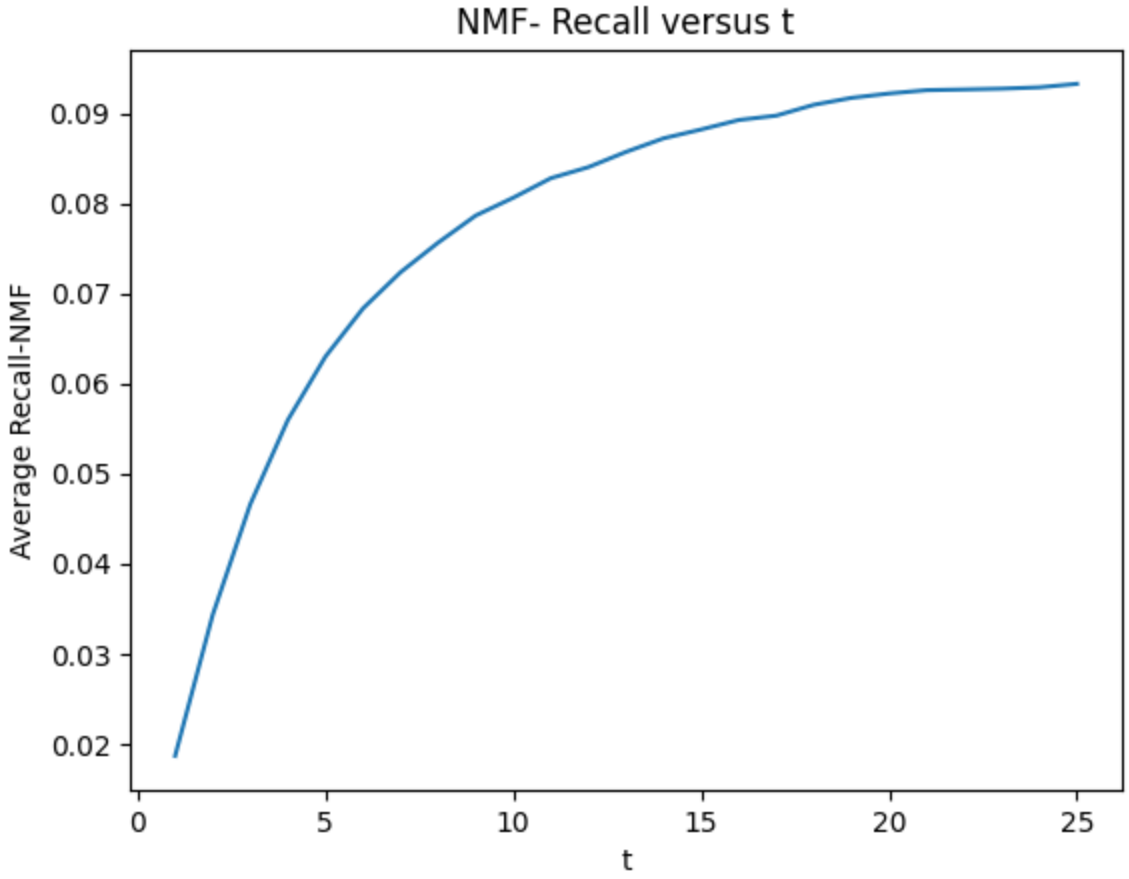
In [110…

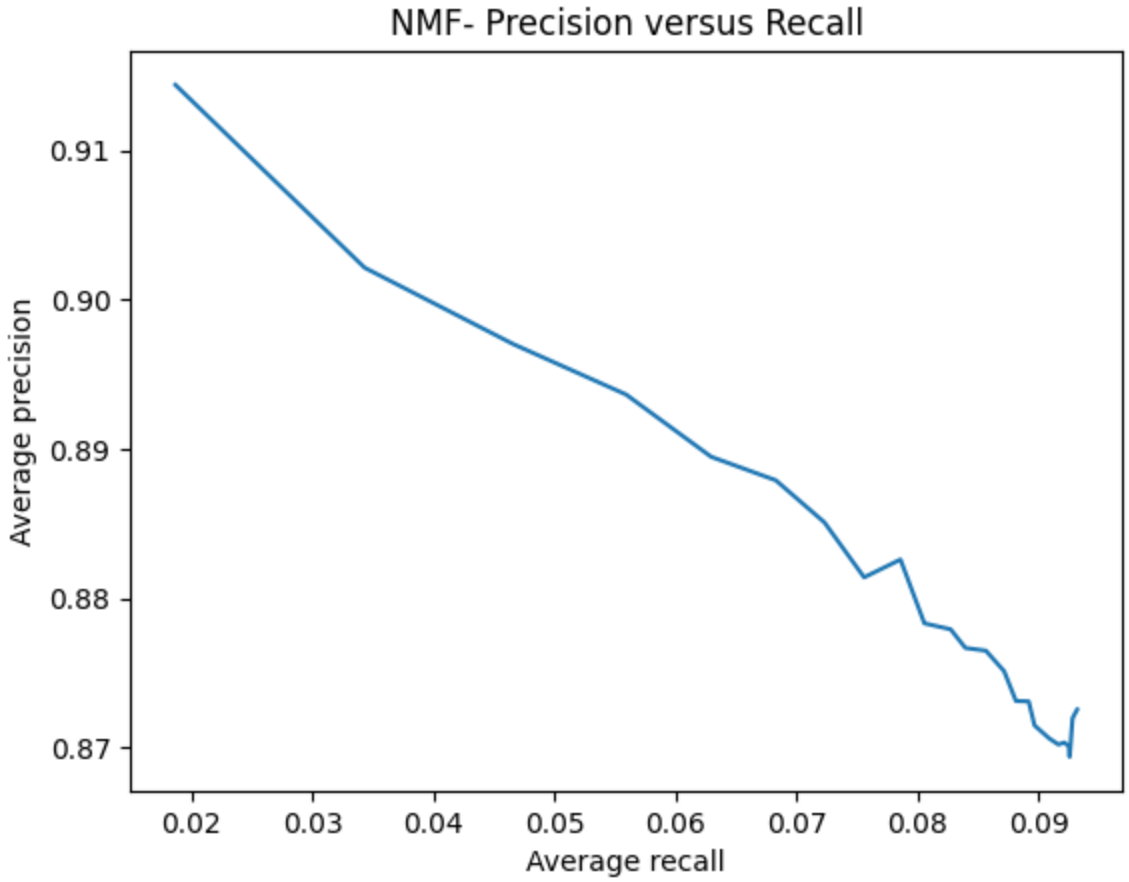
```
pre_nmf, rec_nmf = calculate_precision_recall(ratings, "NMF")
```

```
k=1:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=3:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=5:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```

k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=7:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=9:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=11:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=13:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=15:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=17:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=19:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=21:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=23:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=25:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,





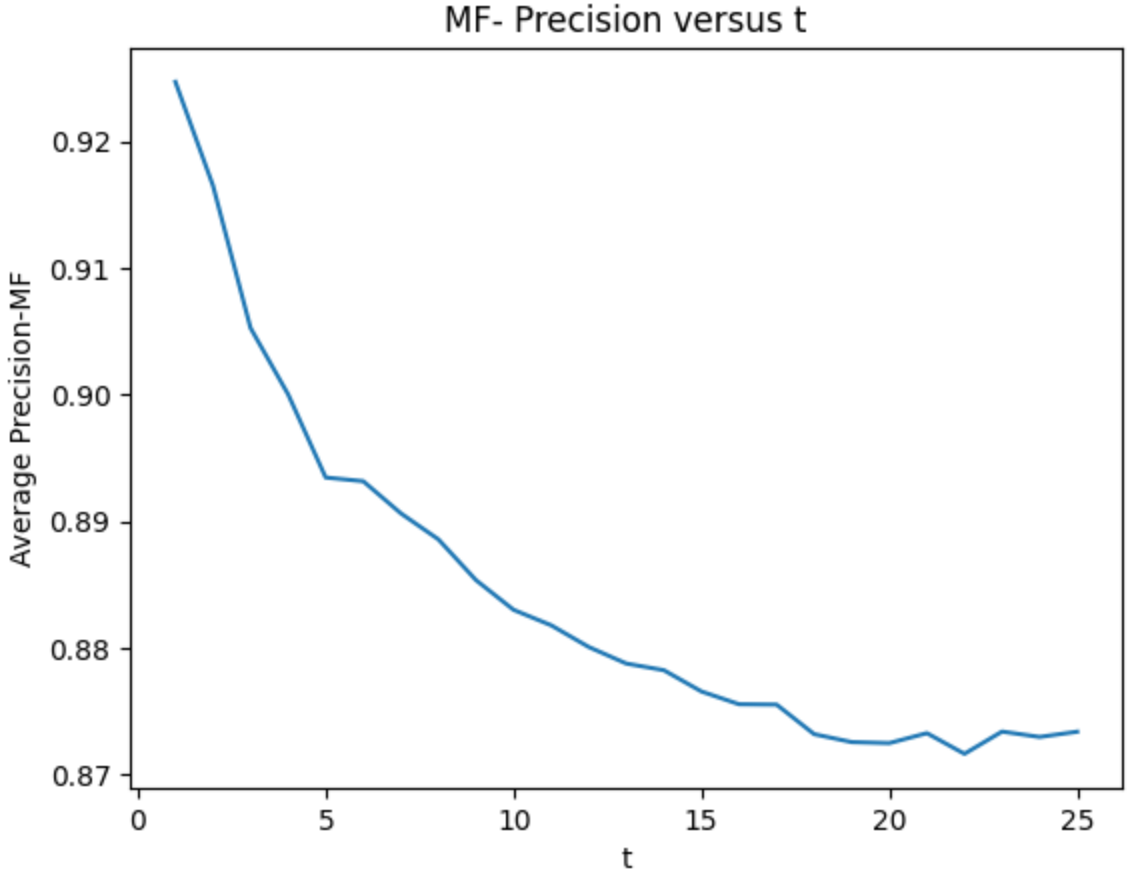


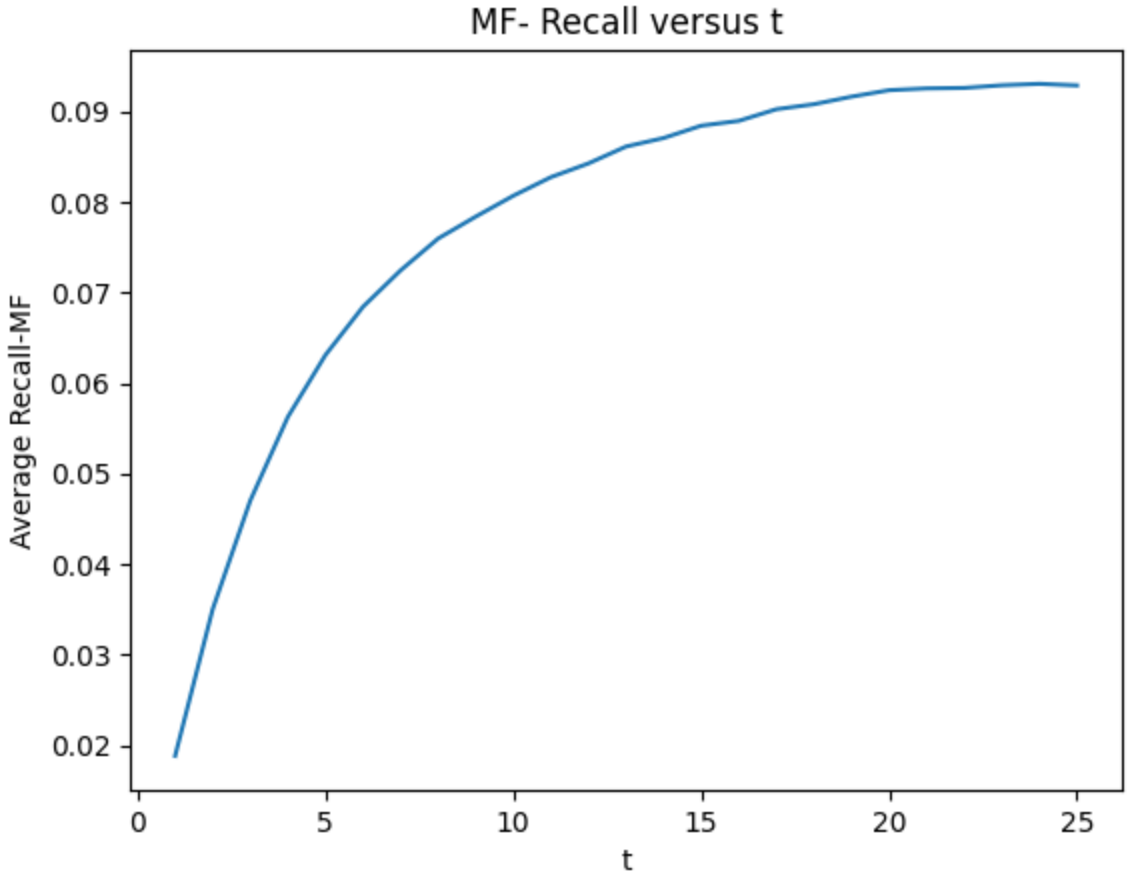
In [111…

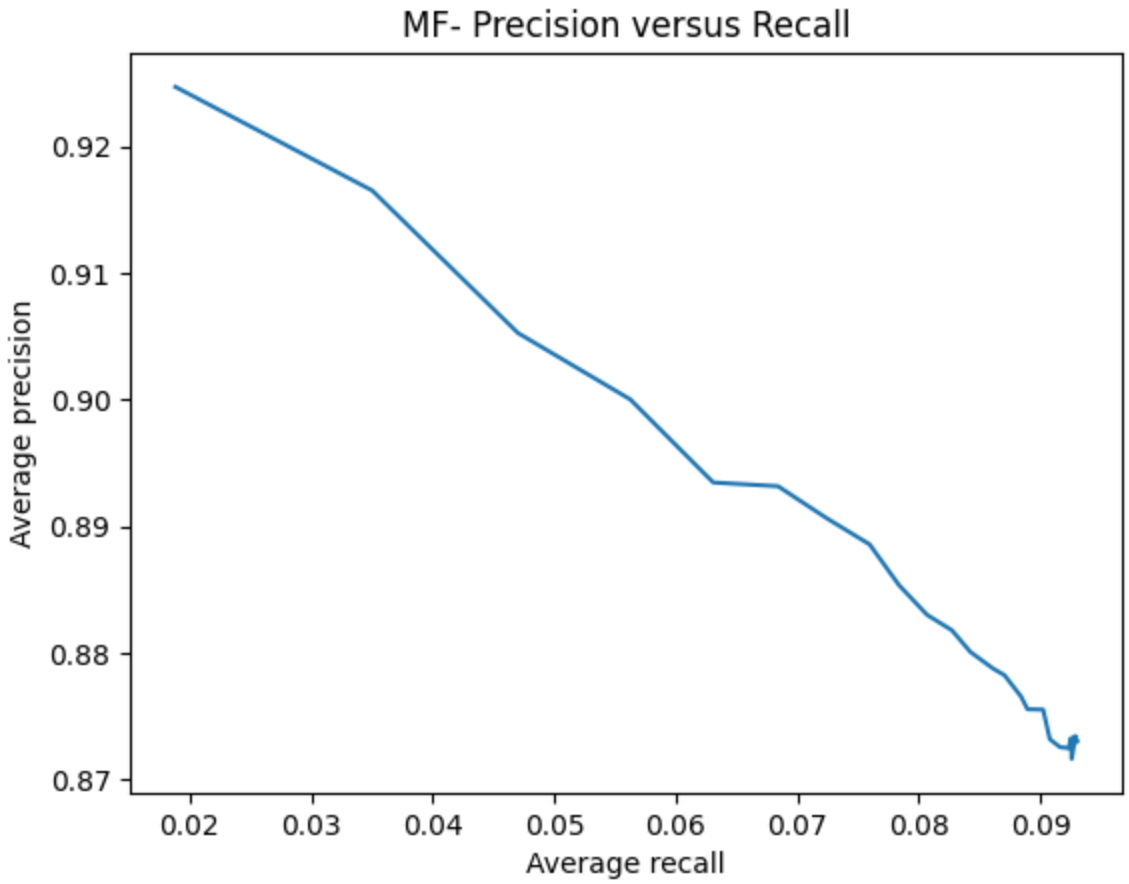
```
pre_mf, rec_mf = calculate_precision_recall(ratings, "MF")
```

```
k=1:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=2:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=3:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=4:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=5:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
```

k=6:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=7:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=8:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=9:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=10:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=11:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=12:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=13:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=14:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=15:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=16:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=17:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=18:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=19:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=20:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=21:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=22:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=23:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=24:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,
k=25:iter=1,iter=2,iter=3,iter=4,iter=5,iter=6,iter=7,iter=8,iter=9,iter=10,







In [112…

```
plt.plot(rec_knn, pre_knn,color='g',label='kNN')
plt.plot(rec_nmf, pre_nmf,color='b',label='NMF')
plt.plot(rec_mf, pre_mf,color='r',label='SVD')

plt.title('Question 14 Comparison of -kNN, NMF, MF - Precision vs Recall')
plt.ylabel('Average precision')
plt.xlabel('Average recall')
```

```
plt.legend(loc="best")
plt.show()
```