Zach Siegel (#805435913), Edward Nguyen (#904663048), Boya (#005037574), Banseok Lee (#605351891)
EE219 - *Large-Scale Data Mining: Models and Algorithms*
Prof. Vwani Roychowdhury
Winter 2021 - Project 3

Question 1.

The rating matrix denoted by R has 1.7% sparsity that means 98.3% of elements are zero in the matrix R.

```
Total possible ratings: 5931640
Number Ratings: 100836
Sparsity: 0.016999683055613623
```

Figure 1: The ratings matrix has one row for each user in the MovieLens `ratings.csv` file and one column for each movie that might be rated from the `movies.csv` file.

Question 2.

As we draw a histogram of movie ratings, the histogram showed asymmetric distribution with a left-skewed form that has a mode rating of 4.0. The left-skewed distribution, however, is not smoothly distributed. All integer ratings; 1.0, 2.0, 3.0, 4.0 showed large frequencies than their related floating ratings; 1.5, 2.5, 3.5, 4.5. One possible reason for it is that our rating is a little bit biased toward nearest integer values when we rating. If we change the binned interval to 1.0, the distribution will be smooth.

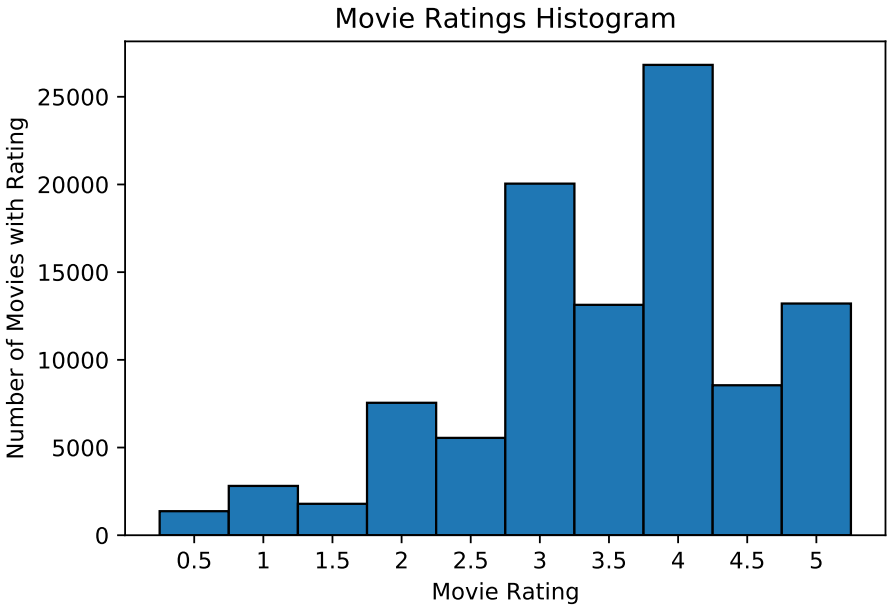
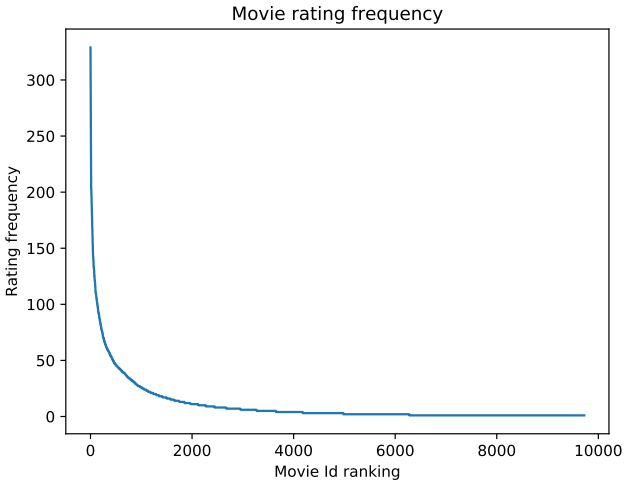
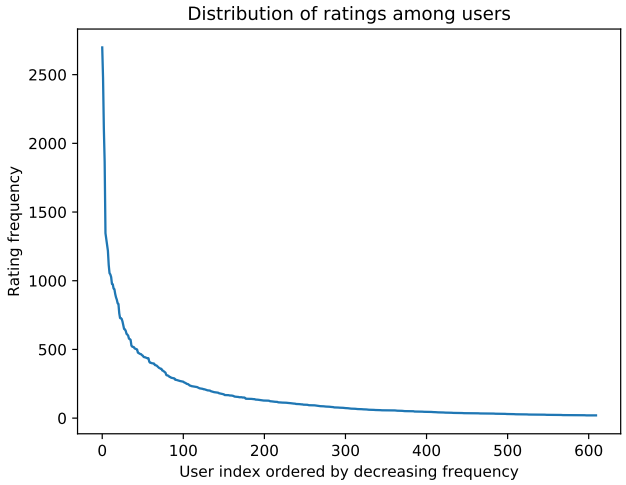


Figure 2: Frequency of the rating values with 0.5 binned intervals.

Questions 3, 4, 5.



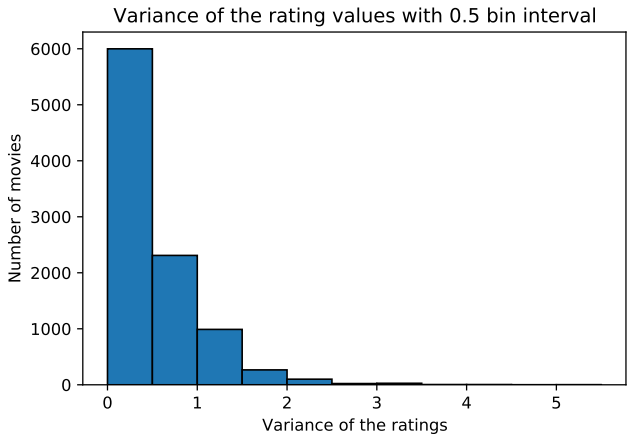
(a) Distribution of the number of ratings received among movies. The X-axis, Movie index ranking is ordered by decreasing frequency.



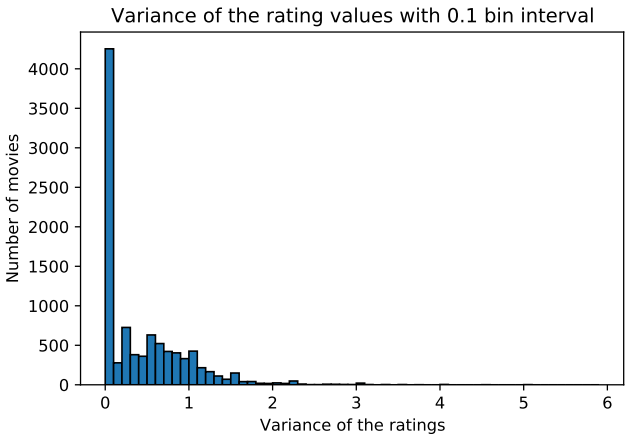
(b) Distribution of the number of ratings received among users. The X-axis, User index ranking is ordered by decreasing frequency.

Clearly, both frequency rating distribution of movies and users are very similar heavy-tale distributions. From their curves, we can see that 20% of movies got 78% of the total ratings, and 20% of users rated 65% of movies. Also, we can analyze a strong correlation between movies with high-frequency and users who frequently give ratings. This analysis implies that it may be possible our collaborative filtering would not be strongly affected whether we discard unpopular movies and some inactive users. Another important thing to note is that very few movies have over 50 ratings. This means that the rating matrix is very sparse. Lastly, there are some movies with very many ratings which seems to insinuate they are very popular.

Question 6.



(a) Histogram with 0.5 bin interval



(b) Histogram 0.1 bin interval

Figure 4: Histogram of the variance of the rating values received by each movie.

The histogram distributions are similar to the shape of exponentially decaying. Almost a third(3744) of the variances in the histogram are zero. Those zeros are from the heavy-tale distribution's tale in the previous problem consists of only one rating value. Another observation is that as most of the variance is less than 0.5 that means that most of the movies are receiving similar ratings from users which means the movies are either universally liked or disliked among the users.

Question 7.

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

$N(I_u)$ is the number of the items in the set of item indices for which ratings have been specified by user u.

Question 8.

$I_u \cap I_v$ is the set of item indices for which ratings have been specified by user v and u. The overlapped indices set is used in the Pearson correlation function. By doing so, we can extract meaningful values from the sparse matrix R.

$I_u \cap I_v = \emptyset$ is possible, and implies that users v and u have not rated any of the same items. This is not only possible, but likely common in a large dataset with many heterogeneous users and many heterogeneous items.

Question 9.

Define a "rating level" as a possible rating; $0.5, 1, 1.5, \ldots, 5$ are those rating levels that appear in the ML dataset. Each user has their own interpretation of a given rating level corresponding to some unobservable utility function, with the only guarantee being that higher ratings correspond to higher utility.

Subtracting each user's mean assumes that the underlying ratings-generating process for each user differs only in their "mean" rating. A more complicated form of centering could be in theory be useful to infer a user's true utility of each movie relative to each other movie. In practice, centering a user's ratings by subtracting their mean rating should reasonably control for user heterogeneity.

For example, if a user rates almost all movies a 5 but rates a given movie a 4, that indicates the user *disliked* the movie, which should *decrease* the predicted rating for that user's k nearest neighbors. On the other hand, for a user who rates most movies less than 3, a 4 indicates they *liked* that movie, which should *increase* the predicted rating for that user's k nearest neighbors. For this reason, subtracting user means from neighbors' predictions is essential in capturing which movies a user's neighbors *did like* and *did not like*.

Question 10.

k-NN collaborative filtering was used to predict ratings of movies in the MovieLens dataset. The full dataset was divided into 10 train-test splits over which test set performance was evaluated and averaged for each value of $k \in \{2, 4, 6, \ldots, 100\}$, where k represents the number of nearest neighbors used for predictions. Using this 10-fold cross-validation, the effect of k on performance can be discerned.

In Figure 5, two accuracy metrics, average root mean-squared error (RMSE) and mean absolute error (MAE), are plotted versus k. Clearly, increasing k has the ability to yield more accurate predictions, but with quickly diminishing returns. There are a total of around 600 users in the full dataset.

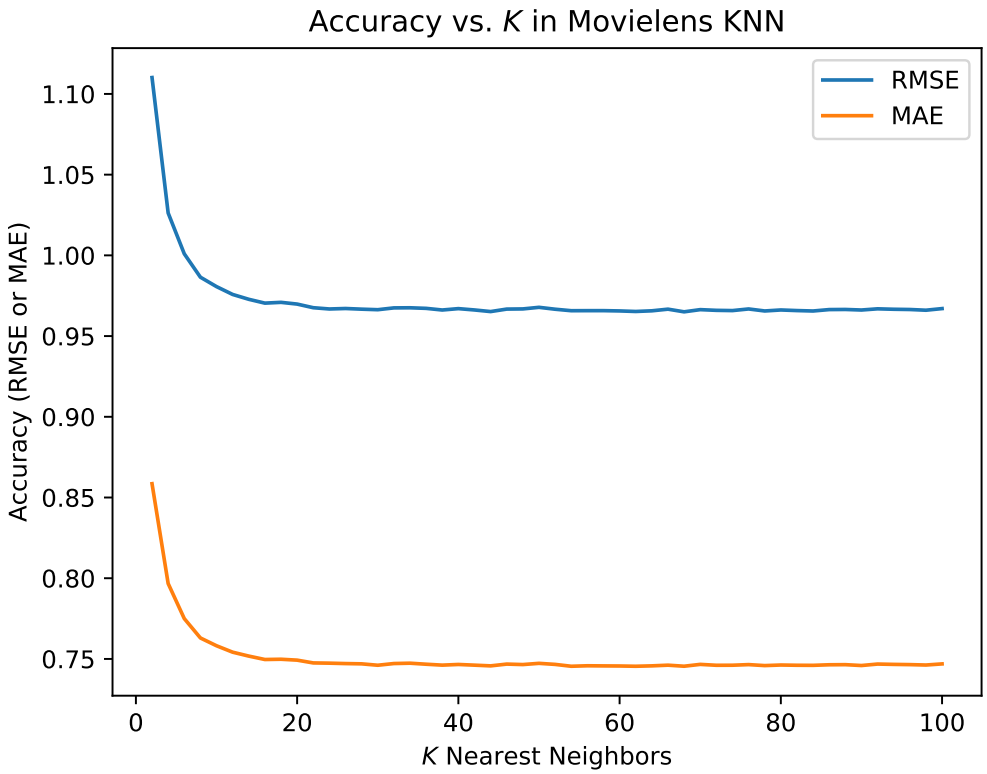


Figure 5: Average accuracy metrics over 10-fold cross-validation are plotted versus number of nearest neighbors k used in a k-NN collaborative filter.

Question 11.

For the remainder of this project, we eyeballed Figure 5 and determined that the value of $k = 20$ is the smallest value of k that yields reasonable performance before the accuracy curve "levels off". The steady state average RMSE value is about 0.975 and the steady state average MAE value is about 0.75.

Questions 12, 13, 14.

As in Figure 5, Figure 6 demonstrates the effect of the number of nearest neighbors k on the accuracy of a k-NN collaborative filter to predict movie ratings. Again, the average test accuracy over 10-fold cross-validation is used to generate each result. Instead of testing on each entire test set, however, each trained k-NN predictor was tested on only a trimmed subset of the test fold.

The left panel of Figure 6 demonstrates the performance on popular items (with more than 2 ratings); the middle panel on unpopular items (with less than or equal to 2 ratings); the right panel on popular movies with high variance (at least 5 ratings and a ratings variance of at least 2).

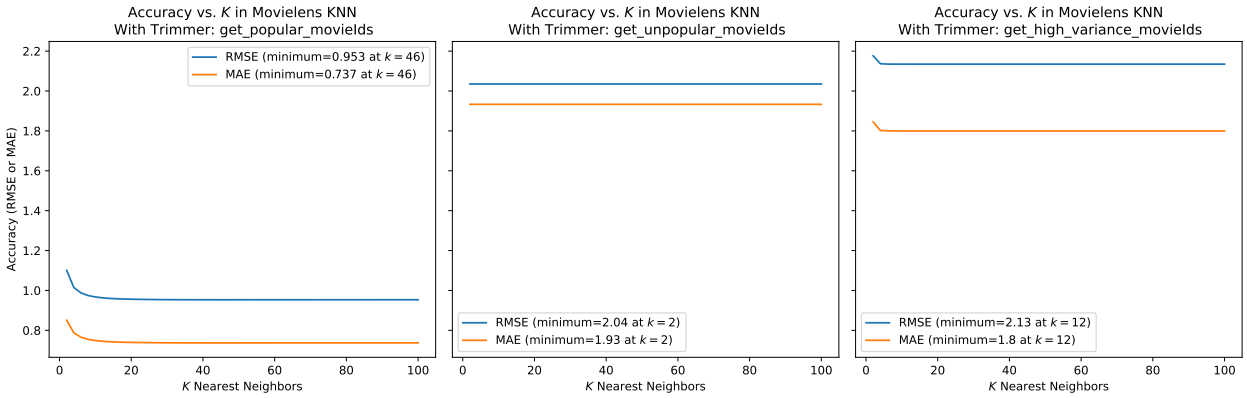


Figure 6: For each trimmed test subset, the accuracy of k-NN collaborative filtering generally improves as k increases, then levels off. Overall, however, performance is worst on unpopular movies and best on popular movies. High-variance popular movies seem to be more difficult to classify than popular movies overall. The minimum average RMSE and MAE for each trimmed test set are reported in the panes.

Question 15.

We determined that the value $k = 20$ yields reasonable performance based on Figure 5. For each threshold in $\{2.5, 3, 3.5, 4\}$, we trained a k-NN classifier using unmodified training data, then determined whether users in the test set "liked" each item by converting their ratings to $\{0, 1\}$ indicating meeting the threshold, and used their predicted ratings to classify them according to this labeling.

Note that in this scheme there are two "thresholds", which we attempt here to disambiguate in the following way: the "rating-threshold" is used to define whether a user "likes" an item, and is in the set $\{2.5, 3, 3.5, 4\}$; the "classification-threshold" is used to define whether a predicted rating should entail a prediction of "liking" an item, and is subject to the relative importance of TPR and FPR.

Assuming a perfect k-NN predictor that exactly predicted the ratings in the test set, there would be a true positive rate (TPR) of 1 and a false positive rate (FPR) of 0 by using a classification-threshold for predictions at precisely the rating-threshold used to convert the test data. However, the prediction is imperfect, and, depending on the relative importance of TPR versus FPR, it may be preferable to use a classification-threshold at a value different from the rating-threshold used to convert the test data.

That is, when seeking to classify users who will "like" items by rating them at rating-threshold τ, it may be preferable to train a classifier and then target only users whose predictions are greater than τ (which would decrease FPR) or less than τ (which would increase FPR). The rating-threshold used to define "liking" an item need not be the preferred classification-threshold used for classification.

Figure 7 demonstrates this tradeoff, by illustrating the TPR and FPR at several classification thresholds along the ROC curve, as well as noting the AUC for the classifier.

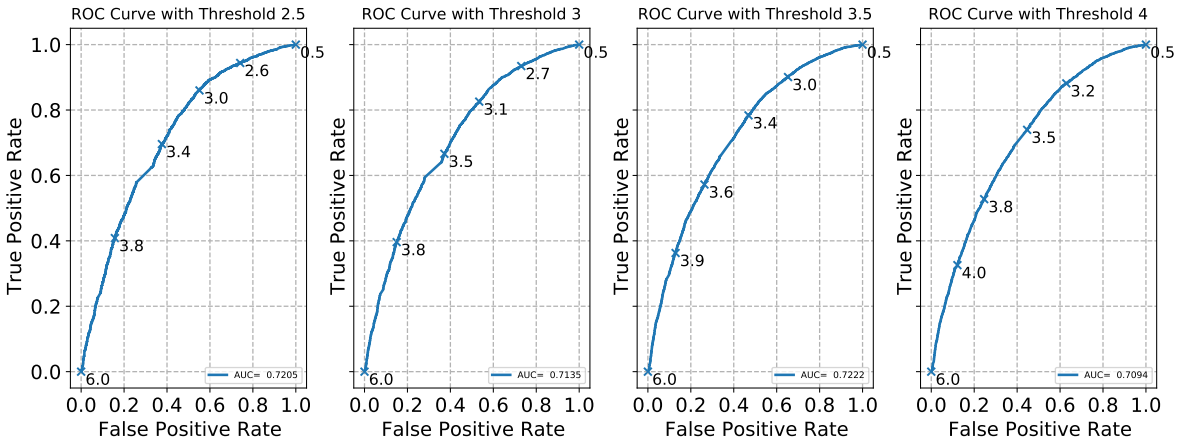


Figure 7: A k-NN collaborative filter was used to predict ratings in a test subset. The "ground truth" for the test observations was considered to be whether the user "liked" the item, i.e. whether their true rating exceeded the rating-threshold. The prediction filter was recast as a classifier, and the classification-thresholds that achieve several levels of TPR and FPR are labeled on the ROC curves, along with the total AUC.

Note that we initially interpreted this question differently! We initially considered thresholding the entire dataset (using a rating-threshold), training a collaborative filtering predictor on the binary training data, and then using that predictor to classify the test set. Binary classification seems like a natural application of collaborative filtering, but **we adjusted our interpretation to that above for three reasons**:

1. The collaborative filtering method utilizes variations in nonbinary ratings, and we felt the question prompt implied that *the same* recommendation system was being used, only with an adjusted test-set evaluation scheme using a binary thresholding.

2. The NMF prediction method in the `Surprise` package (introduced later on) simply does not support training datasets in which items have no reviews or reviews that are all zero. If the full dataset is thresholded before training, particularly for high thresholds, some items in the training set will have only a combination of zero ratings and no ratings, and this function will fail (as it did for us several times, depending on the train-test split). This suggested to us that the techniques we are practicing are inappropriate for training on binary datasets.

3. We started a discussion on the class' Piazza forum about this, and two students from other groups mentioned that they did not threshold the entire dataset. We reached out to the TA who confirmed the intention of this question. We feel that this is a matter of interpretation and did the work both ways, and we hope that other groups who shared our initial interpretation do not lose points.

Question 16.

The optimization problem given by equation 5 $\min_{U,V} \sum_{i=1}^{m} \sum_{j=1}^{n} W_{ij}(r_{ij} - (UV^T)_{ij})^2$ is not convex. This is because of the fact that we are minimizing over U and V which would involve the term $u_i \dot{v}_j^T$

which is not convex. If we were to fix U, then the problem is convex and we can formulate it as a least squares problem.

$$\min_V \sum_{i=1}^{m}\sum_{j=1}^{n} W_{ij}(r_{ij} - (UV^T)_{ij})^2 \tag{1}$$

$$\approx \min_V \sum_{i=1}^{m}\sum_{j=1}^{n} (W_{ij}r_{ij} - W_{ij}(UV^T)_{ij})^2 \tag{2}$$

Define matrix A as a $mn \times nk$ matrix. Define vector b as vector of size mn. Reshape matrix V into a vector of size nk. Redefine matrix U^T as $[u_1, u_2, ..., u_m]$ where u_i is a column of U^T. Then U would be

$$\begin{bmatrix} u_1^T \\ \vdots \\ u_m^T \end{bmatrix}$$

Then matrix A would be as follows

$$A = \begin{bmatrix} W_{11}u_1^T & 0 & \cdots & \cdots & 0 \\ \vdots & W_{12}u_1^T & \cdots & \cdots & \vdots \\ 0 & \cdots & \cdots & \cdots & W_{1n}u_1^T \\ W_{21}u_2^T & 0 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & \cdots & W_{mn}u_m^T \end{bmatrix} \tag{3}$$

Vector b would be as follows

$$b = \begin{bmatrix} W_{11}r_{11} \\ W_{12}r_{r12} \\ \vdots \\ W_{21}r_{21} \\ \vdots \\ W_{mn}r_{mn} \end{bmatrix} \tag{4}$$

To see the reshaped matrix we first redefine $V^T = [v_1, \cdots, v_n]$. The reshaped matrix V we will denote as $\hat{v}$ and is defined as follows

$$\hat{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \tag{5}$$

With this reformulation we now have the least squares problem as follows:

$$\min_{\hat{v}} \|A\hat{v} - b\|^2 \tag{6}$$

Question 17.

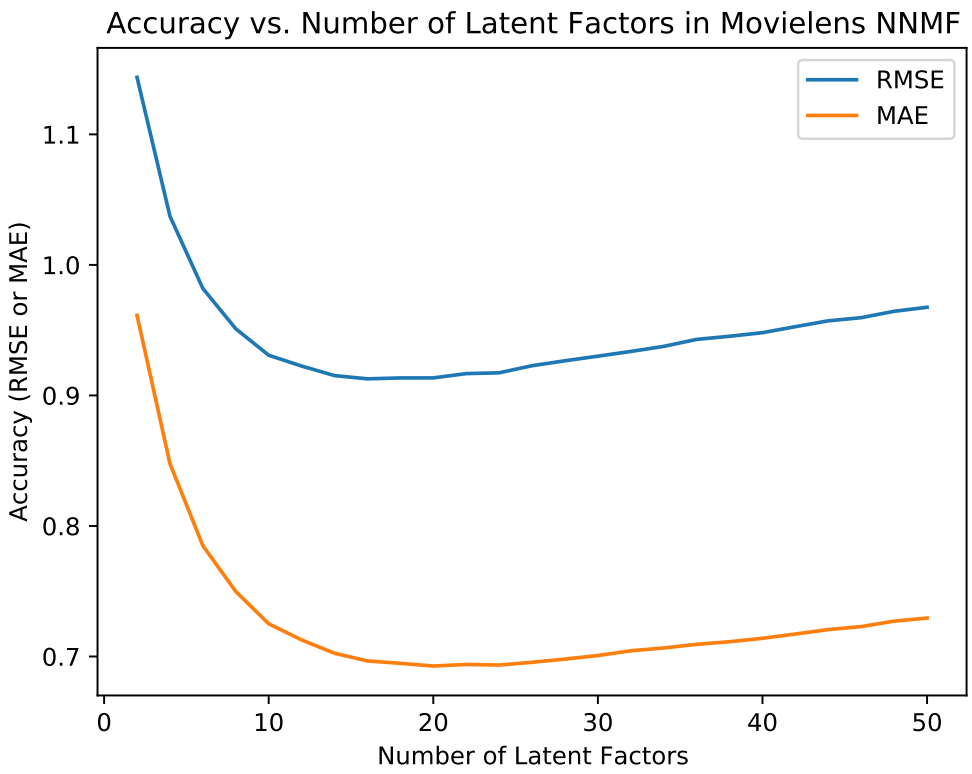


Figure 8: Similar to Figure 5, average accuracy metrics over 10-fold cross-validation are plotted versus number of latent factors used in a NNMF collaborative filter.

Question 18.

The optimal number of latent factors is 20. The minimum average RMSE value is 0.9127. The minimum average MAE value is 0.6926. There are 18 different genres so while the optimal number of latent factors is not exactly the same as the number of movie genres, it is close.

Questions 19, 20, 21.

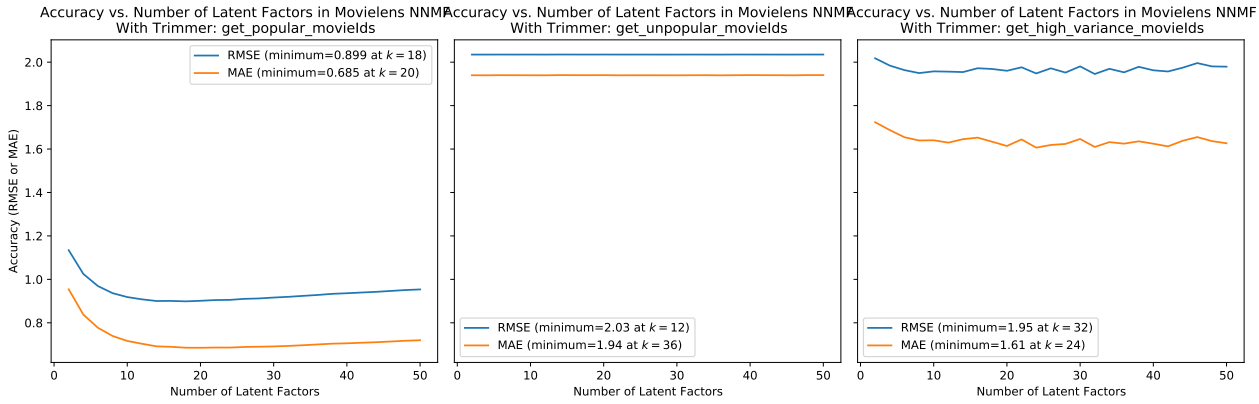


Figure 9: Similar to Figure 6, for each trimmed test subset, the accuracy of NNMF collaborative filtering generally improves as the number of latent factors increases, then levels off. Overall, however, performance is worst on unpopular movies and best on popular movies. High-variance popular movies seem to be more difficult to classify than popular movies overall. The minimum average RMSE and MAE for each trimmed test set are reported in the panes.

Question 22.

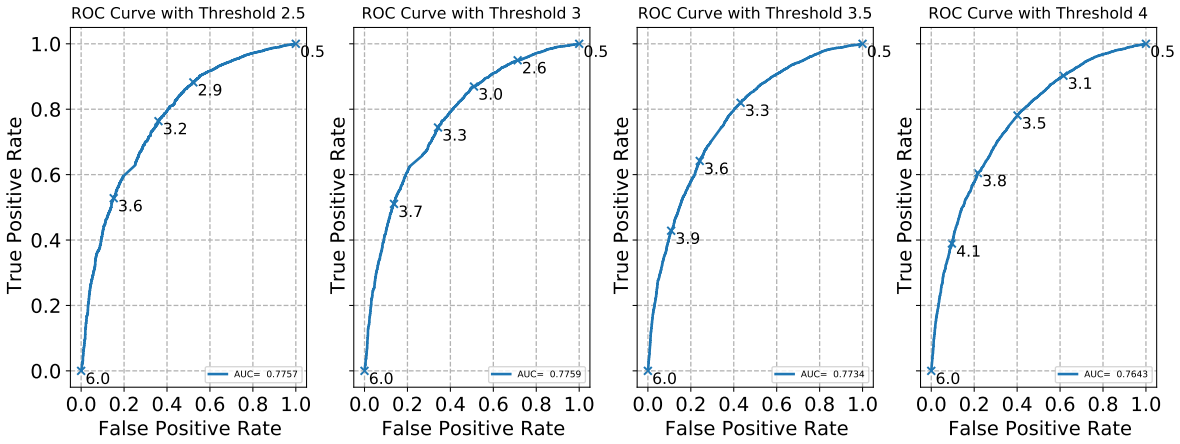


Figure 10: Similar to Figure 7, a NNMF-based collaborative filter was used to predict ratings in a test subset. The "ground truth" for the test observations was considered to be whether the user "liked" the item, i.e. whether their true rating exceeded the rating-threshold. The prediction filter was recast as a classifier, and the classification-thresholds that achieve several levels of TPR and FPR are labeled on the ROC curves, along with the total AUC.

Question 23.

We performed 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. For each column of V, we sorted the movies in descending. We report the genres of the top 10 movies for some columns:

Column 0	Column 4
Adventure—Animation—Children—Comedy—Fantasy	Action
Comedy—Romance	Drama—Romance
Comedy—Western	Crime—Drama
Comedy—Romance	Animation—Children—Comedy
Drama	Action—Crime—Drama—Thriller
Comedy	Action—Crime—Drama
Comedy—Horror—Sci-Fi	Comedy—Drama
Comedy—Drama—Romance	Drama
Drama	Comedy—Drama
Action—Animation	Action—Comedy—Sci-Fi

Column 6	Column 14
Drama	Animation—Comedy—Drama—Fantasy
Comedy—Romance	Horror
Comedy—Sci-Fi	Comedy
Comedy—Romance	Drama—Musical
Action—Comedy—Crime—Drama—Horror—Thriller	Comedy—Drama—Fantasy—Romance
Action—Comedy—Horror—Sci-Fi	Crime
Comedy—Horror—Sci-Fi	Comedy—Romance
Comedy	Comedy
Action—Horror—Sci-Fi—Thriller	Comedy—Drama
Adventure—Comedy—Sci-Fi	Horror

In Column 0 and Column 6 we can see that most of the movies belong to the comedy genre. In Column 4, most of the movies belong to the drama genre. In Column 14, most of the movies belong to the comedy and drama genre. Thus, the top 10 movies seem to belong to a particular or a small collection of genres and seems to suggest that there is a strong connection between the latent factors and the movie genres. Each column (20 columns for each latent factor) seems to correspond to a particular genre of movie as each column is a group of movies of a similar genre.

Question 24.

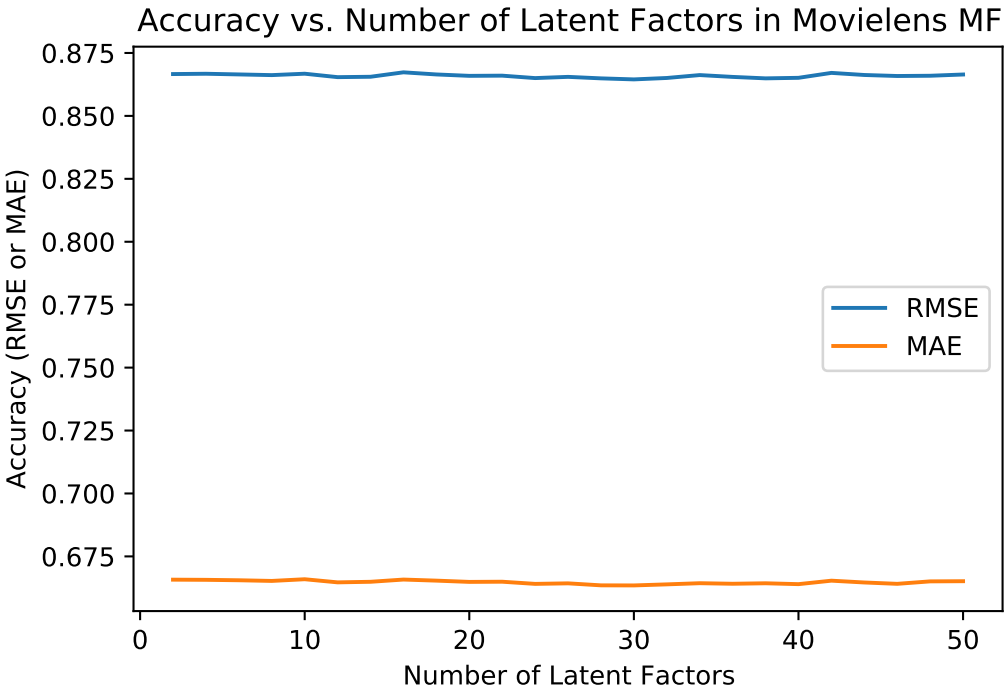


Figure 11: Similar to Figure 5, average accuracy metrics over 10-fold cross-validation are plotted versus number of latent factors used in a MF collaborative filter.

Question 25.

Although from the numeric outcome, optimal K is 30, the accuracy RMSE and MAE difference between maximum and minimum is less than 1 percent. Thus with only these outcomes, it is uncertain whether the k value is true optimal. The minimum average RMSE is 0.865. The minimum average MAE is 0.665.

Questions 26, 27, 28.

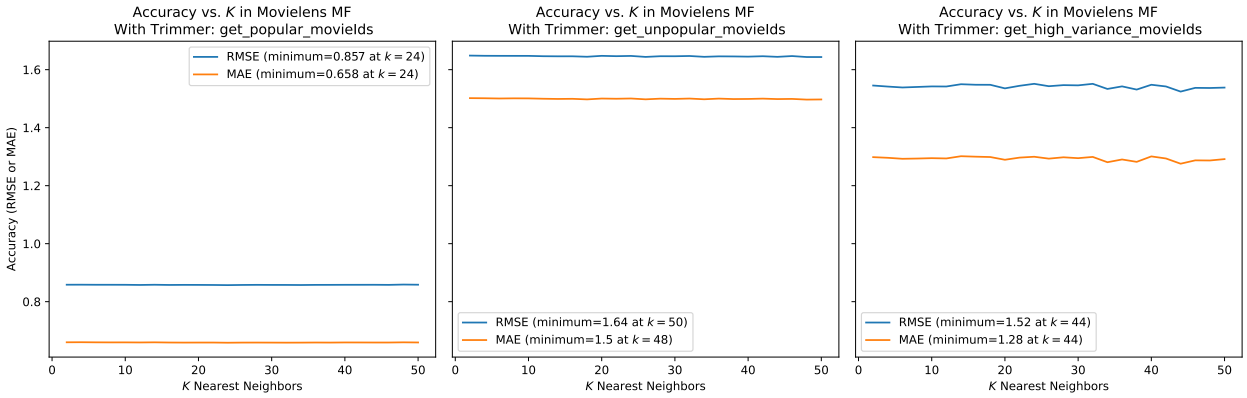


Figure 12: Similar to Figure 6, for each trimmed test subset, the accuracy of a MF collaborative filtering generally improves as k increases, then levels off. Overall, however, performance is worst on unpopular movies and best on popular movies. High-variance popular movies seem to be more difficult to classify than popular movies overall. The minimum average RMSE and MAE for each trimmed test set are reported in the panes.

Question 29.

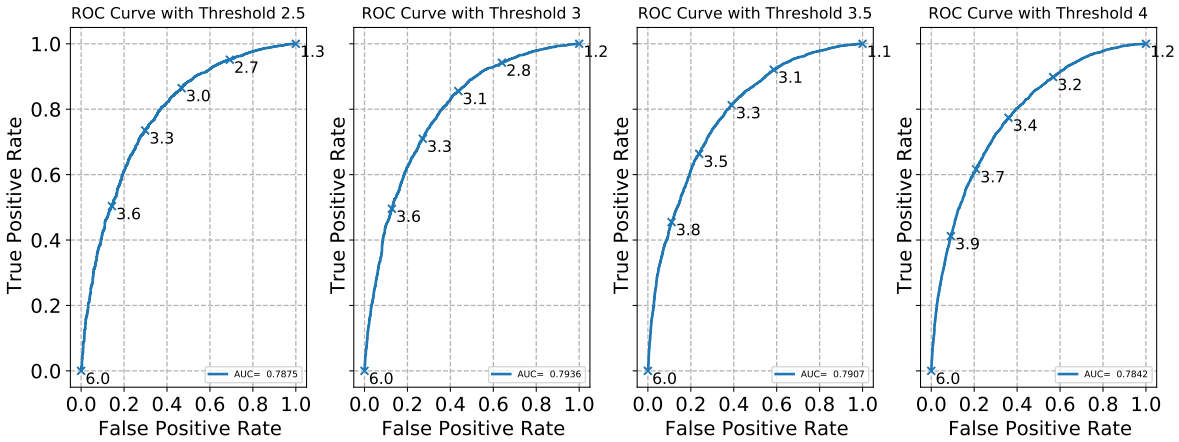


Figure 13: Similar to Figure 7, a MF with bias collaborative filter was used to predict ratings in a test subset. The "ground truth" for the test observations was considered to be whether the user "liked" the item, i.e. whether their true rating exceeded the rating-threshold. The prediction filter was recast as a classifier, and the classification-thresholds that achieve several levels of TPR and FPR are labeled on the ROC curves, along with the total AUC.

Questions 30, 31, 32, 33.

The average RMSE by averaging the RMSE across 10 folds for all data set : 0.9657

The average RMSE by averaging the RMSE across 10 folds for the popular trimmed set: 0.9554

The average RMSE by averaging the RMSE across 10 folds for the unpopular trimmed set: 1.8906
The average RMSE by averaging the RMSE across 10 folds for the high variance trimmed set: 1.5948

Question 34.

In former problems, we set up the four different classifiers: k-NN, Non-Negative MF, MF with bias, and Naive collaborative filters. We've got all scores, and to compare them, draw the ROC curves with threshold 3 for the same dataset. By comparing the AUC value from the ROC curve, we could obtain the ranking; First: MF with bias, Second: NNMF, Third: Naive CF, and Fourth: k-NN classifiers. The first and second, MF with bias and NNMF are similar classifiers except for the existence of bias terms. Both AUC scores have a higher gap between the k-NN and Naive CF scores. This implies that the Matrix factorization-based algorithms are quite well fit for the MovieLens dataset. The weird thing is for the third and fourth score. Naive CF just consists of the average rating of the items for each user but it showed a higher AUC score than the k-NN classifier. It can be seen as the k-NN method is not proper for classifying our dataset.

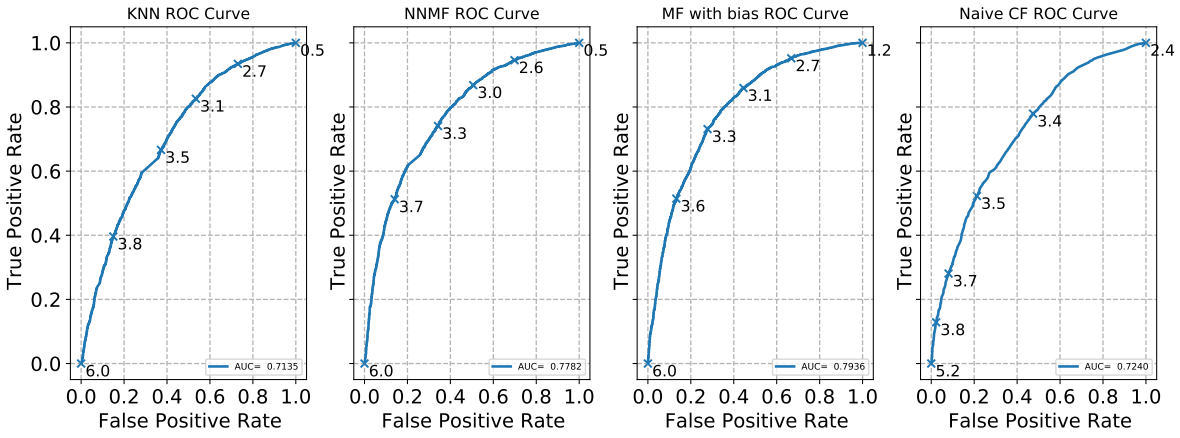


Figure 14: k-NN, Non-Negative MF, MF with bias, and Naive collaborative filters were used to predict ratings in a test subset. The "ground truth" for the test observations was considered to be whether the user "liked" the item, i.e. whether their true rating exceeded the rating-threshold 3. The prediction filter was recast as a classifier, and the classification-threshold that achieves several levels of TPR and FPR are labeled on the ROC curves, along with the total AUC.

Question 35.

Precision can be seen as a measure of quality, and recall as a measure of quantity. Higher precision means that an algorithm returns more relevant results than irrelevant ones, and high recall means that an algorithm returns most of the relevant results.

To be more precise, precision is the fraction of correct positive predictions (true positives) over the total number of predicted positive examples (true positives plus false positives). When the precision is

high, this means that we have a low false positive rate. When the precision is low, this means we have a high false positive rate. It gives us a measure of how many selected items are relevant.

Recall is the fraction of correct positive predictions (true positives) over the relevant elements (true positives plus false negatives). When the recall is high, the amount of relevant instances that were retrieved is high.

Question 36.

The precision vs t, recall vs t and precision vs recall curve for Knn is shown as figure 15, 16 ,17. The precision decreases with t while recall increases with t. This trend makes sense because the more movies that we recommend to the user will give us a greater chance of recommending something the user will like. However, this does lower the chance of everything that we recommend being something that the user likes. There is a clear trade-off between precision and recall that we see.

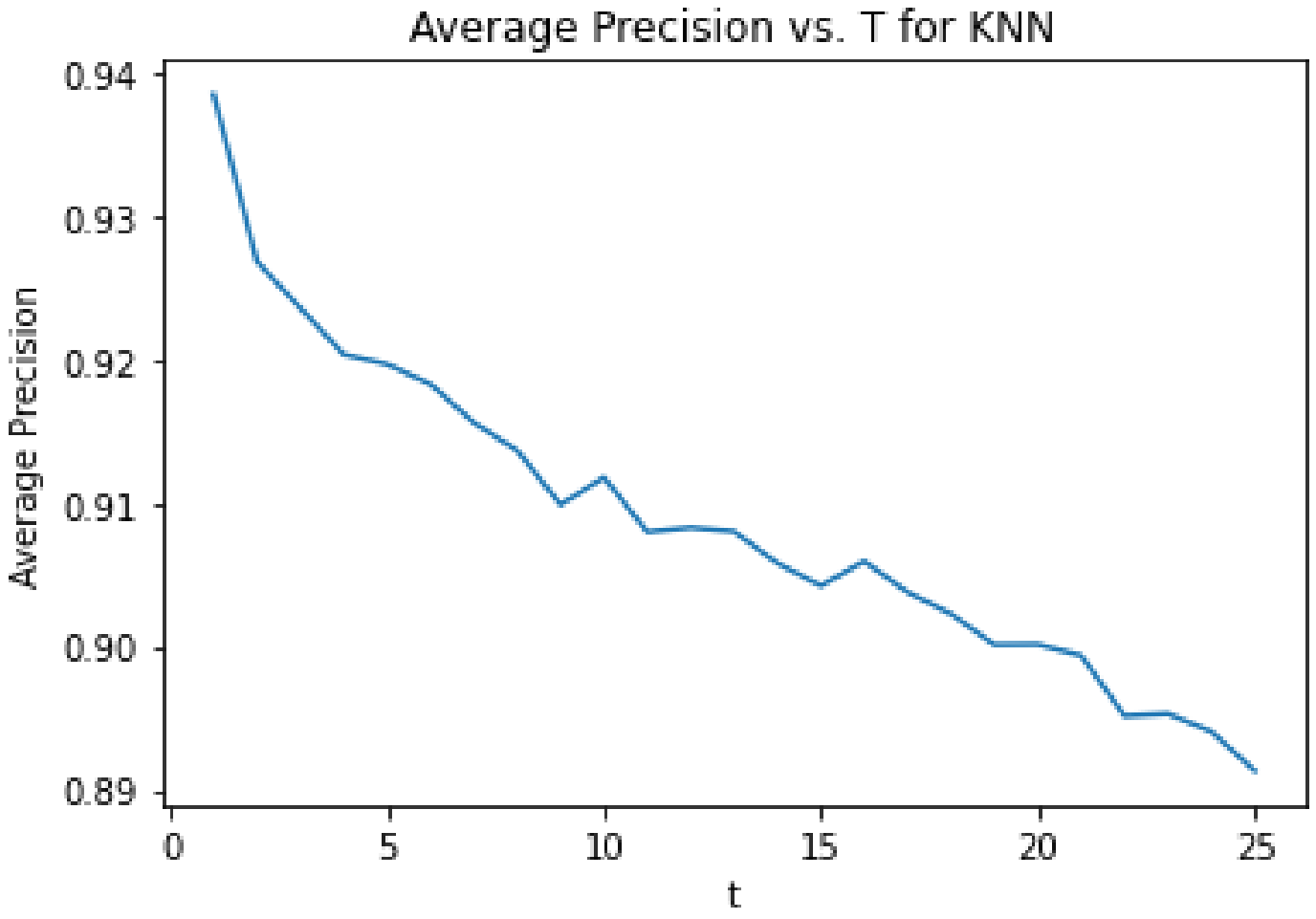


Figure 15: Average precision as function of t for k-NN collborative filter predictions.

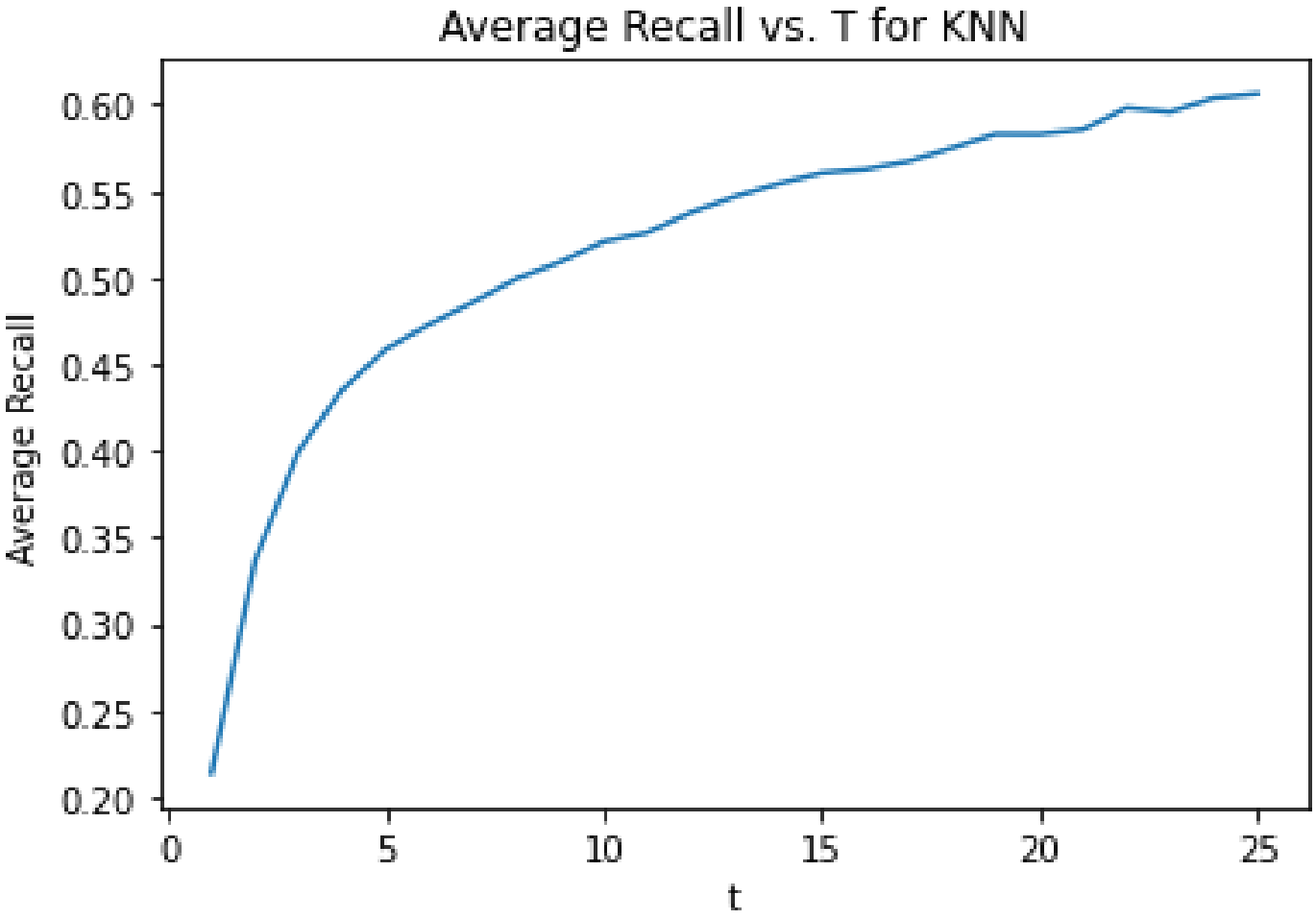


Figure 16: Average recall as function of t for k-NN collborative filter predictions.

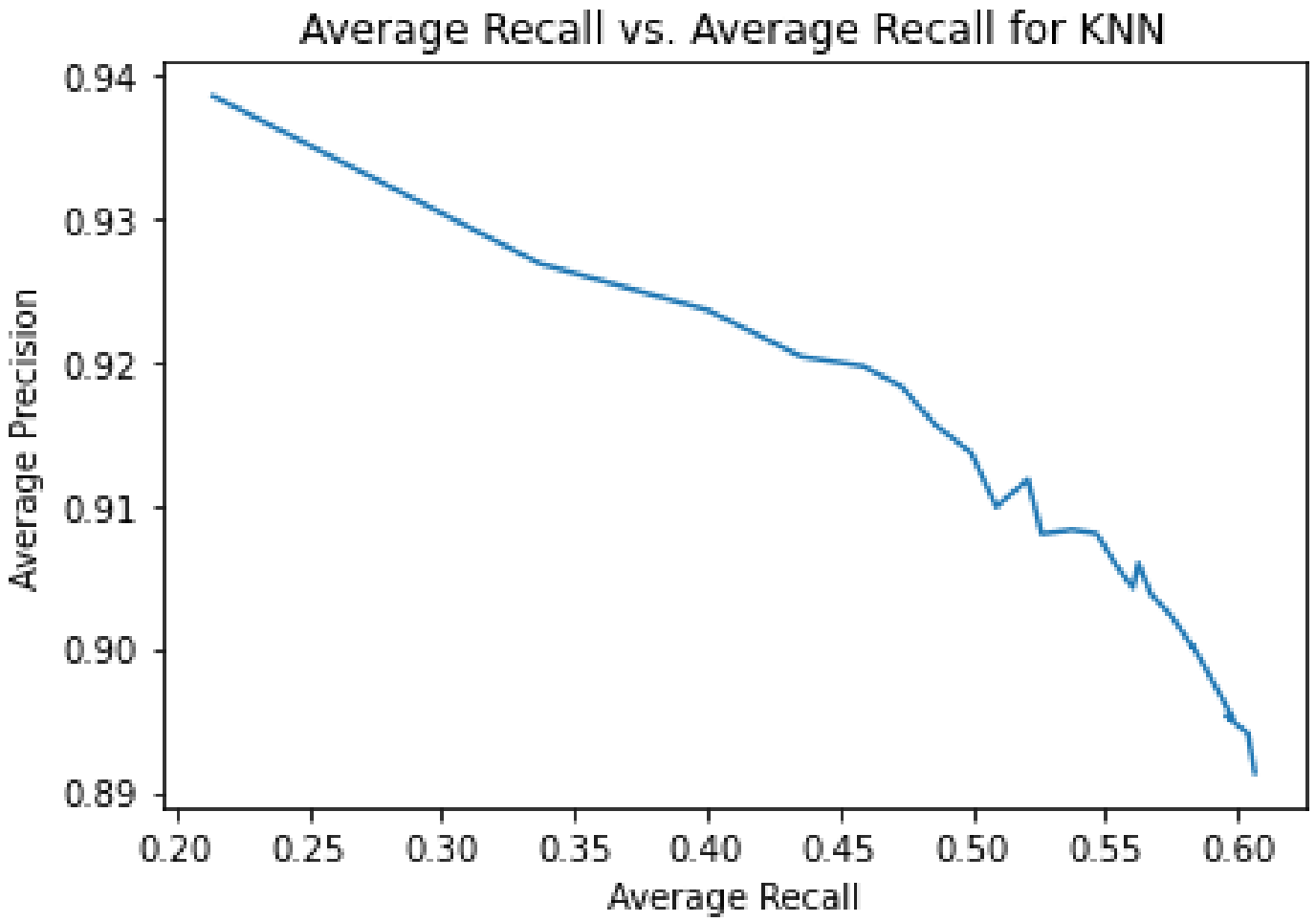


Figure 17: Average precision as function of average recall for k-NN collborative filter predictions.

Question 37.

The precision vs t, recall vs t and precision vs recall curve for NNMF is shown as figure 18, 19 ,20. The precision decreases with t while recall increases with t. This trend makes sense because the more movies that we recommend to the user will give us a greater chance of recommending something the user will like. However, this does lower the chance of everything that we recommend being something that the user likes. There is a clear trade-off between precision and recall that we see.

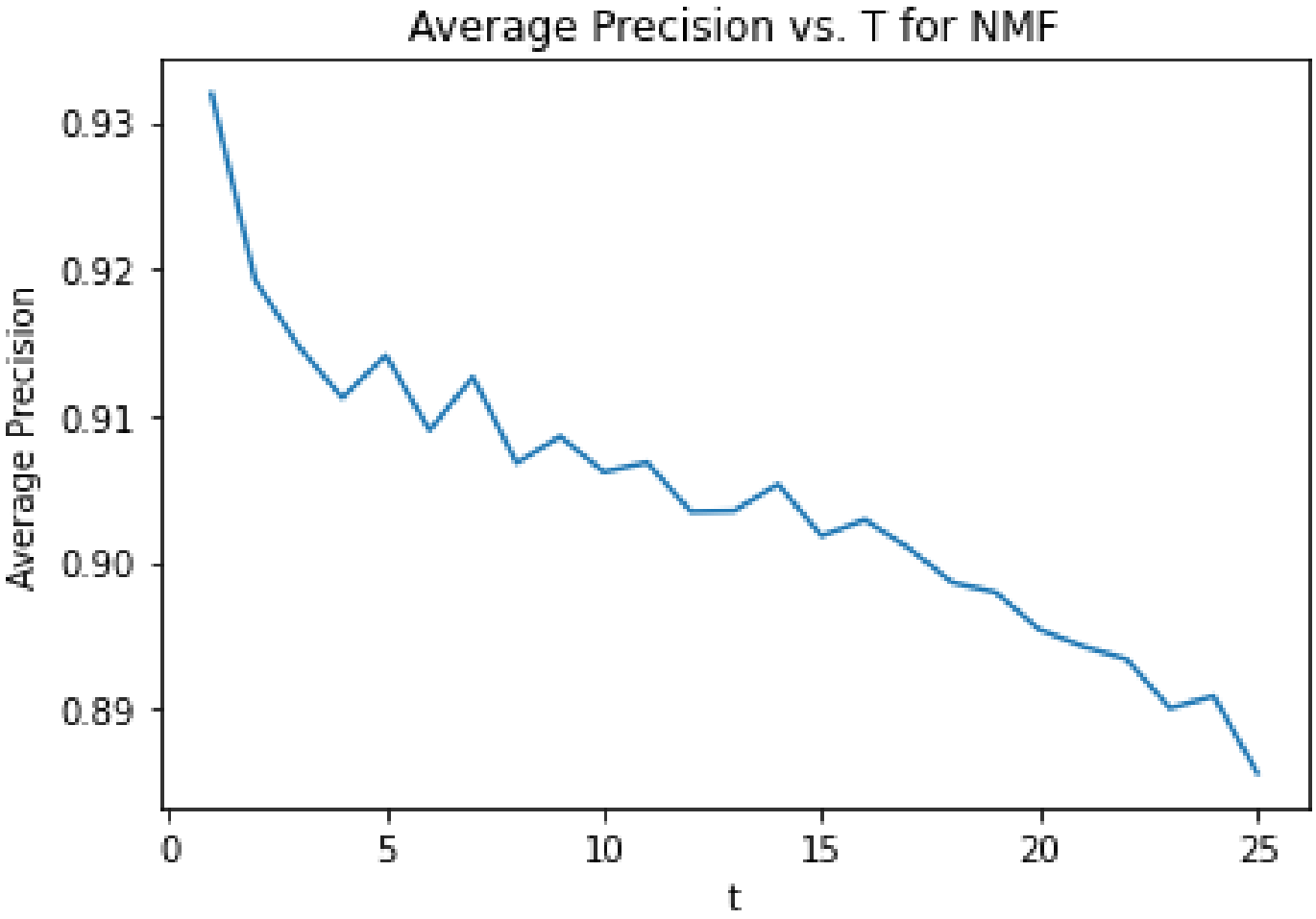


Figure 18: Average precision as function of t for NNMF filter predictions.

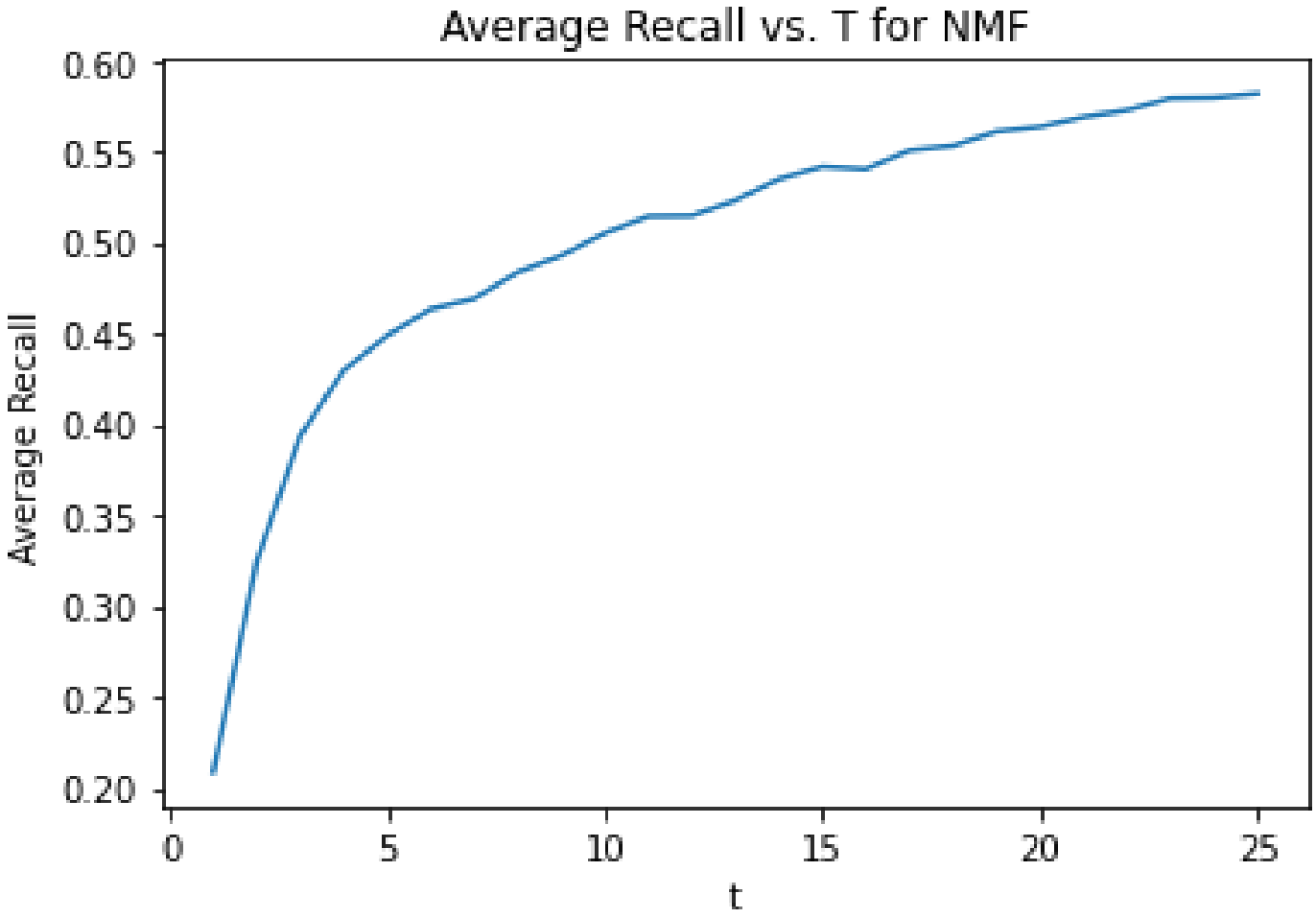


Figure 19: Average recall as function of t for NNMF filter predictions.

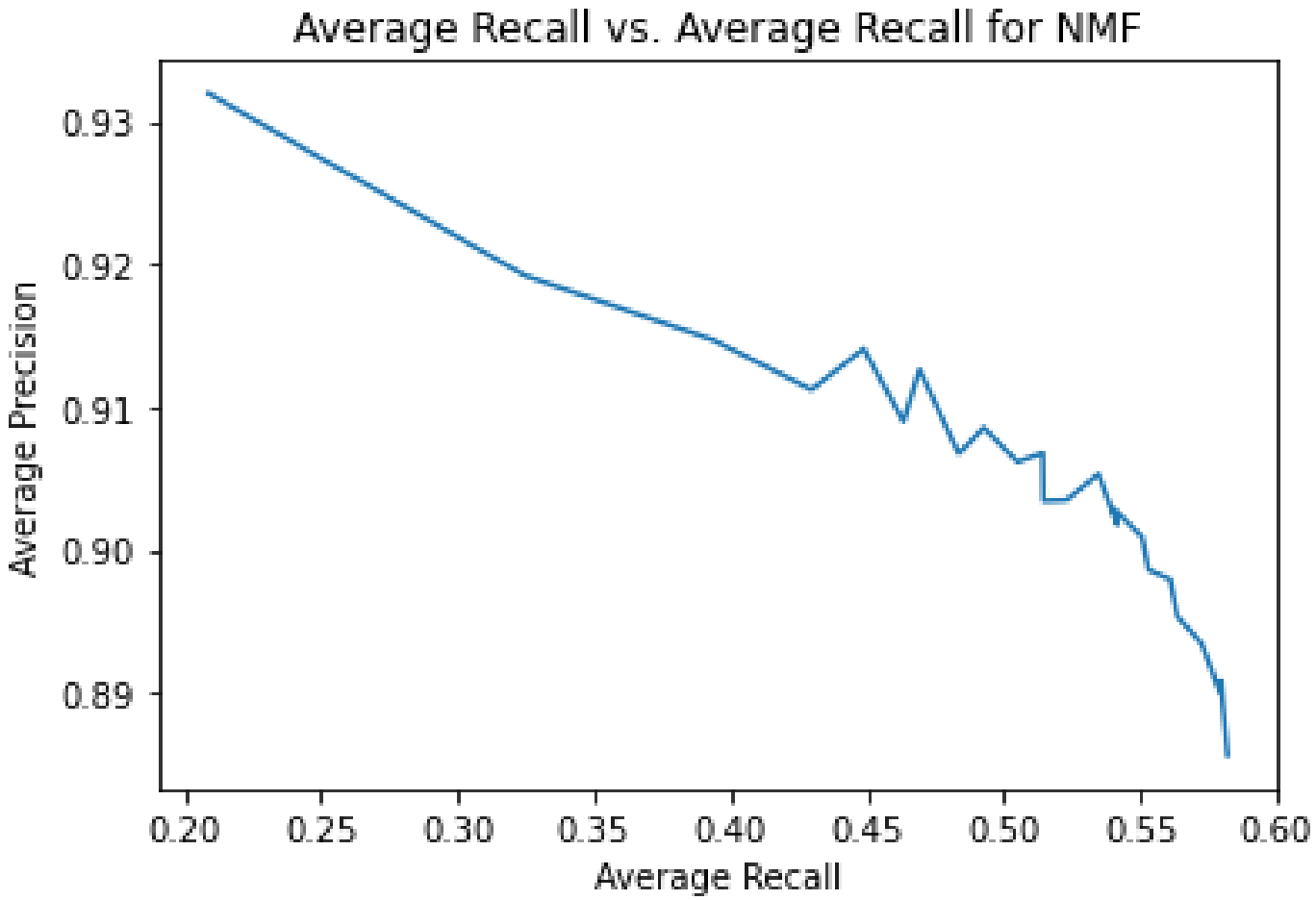


Figure 20: Average precision as function of average recall for NNMF filter predictions.

Question 38.

The precision vs t, recall vs t and precision vs recall curve for Knn is shown as figure 21, 22 ,23. The precision decreases with t while recall increases with t. This trend makes sense because the more movies that we recommend to the user will give us a greater chance of recommending something the user will like. However, this does lower the chance of everything that we recommend being something that the user likes. There is a clear trade-off between precision and recall that we see.

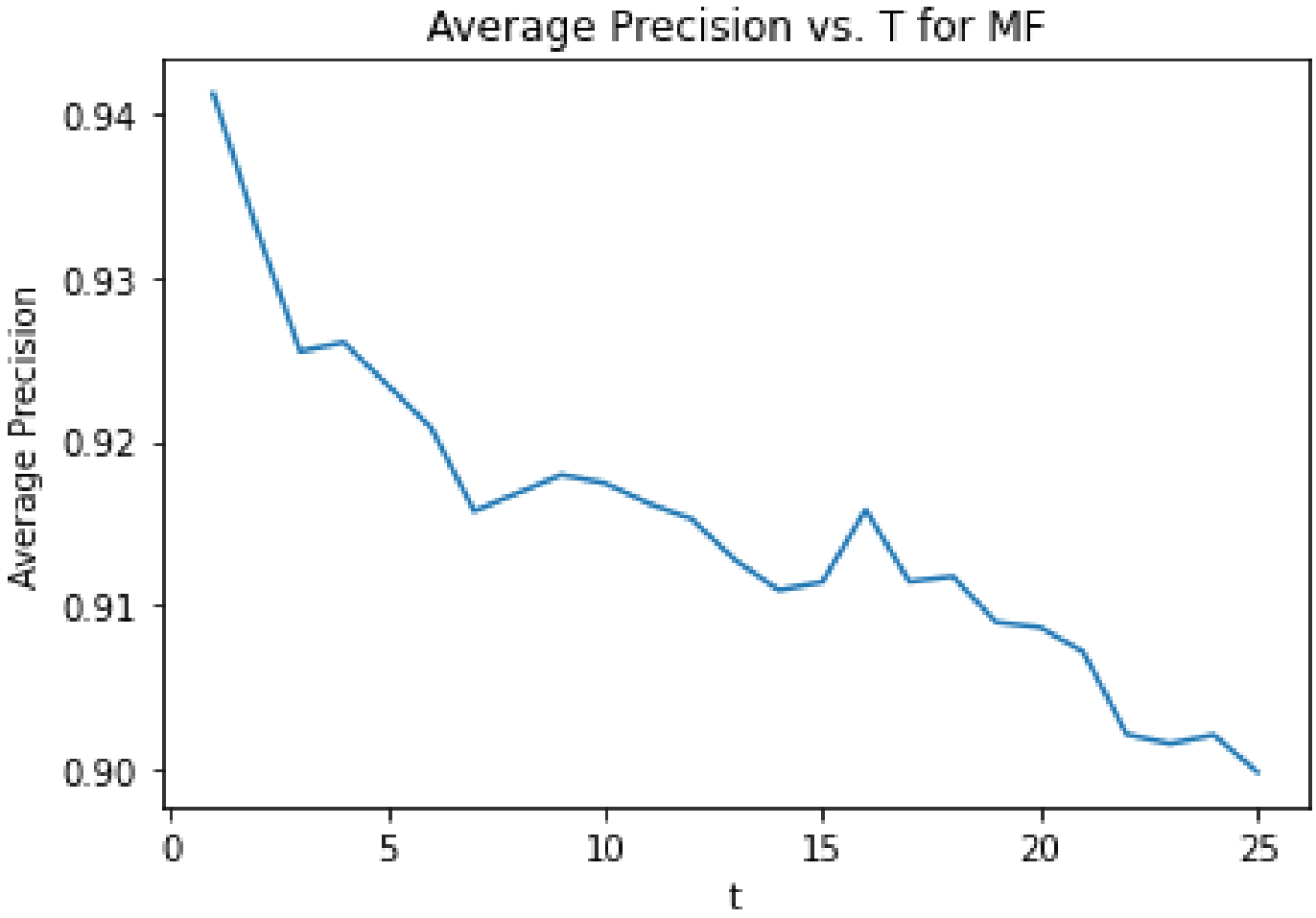


Figure 21: Average precision as function of t for MF filter predictions.

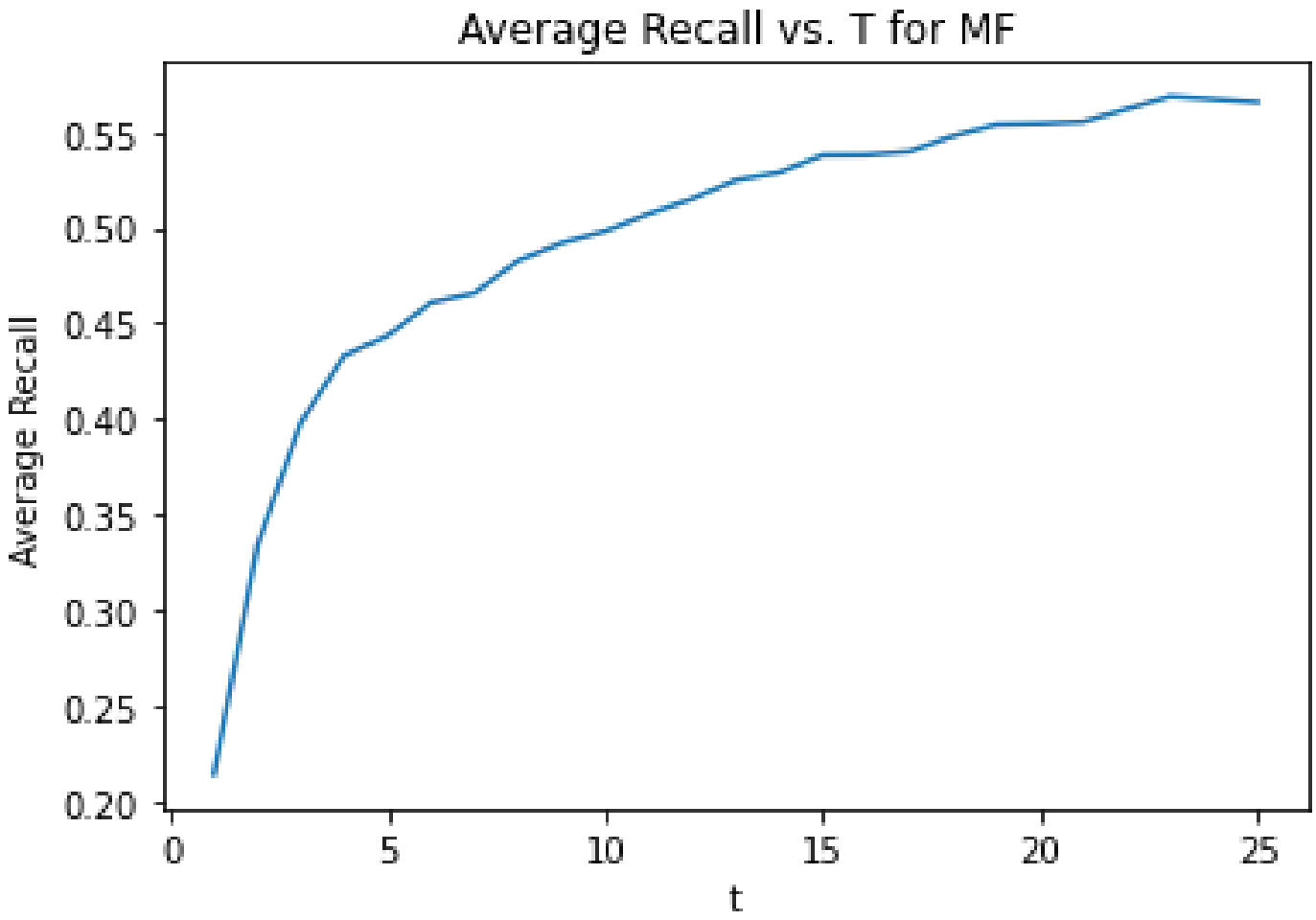


Figure 22: Average recall as function of t for MF filter predictions.

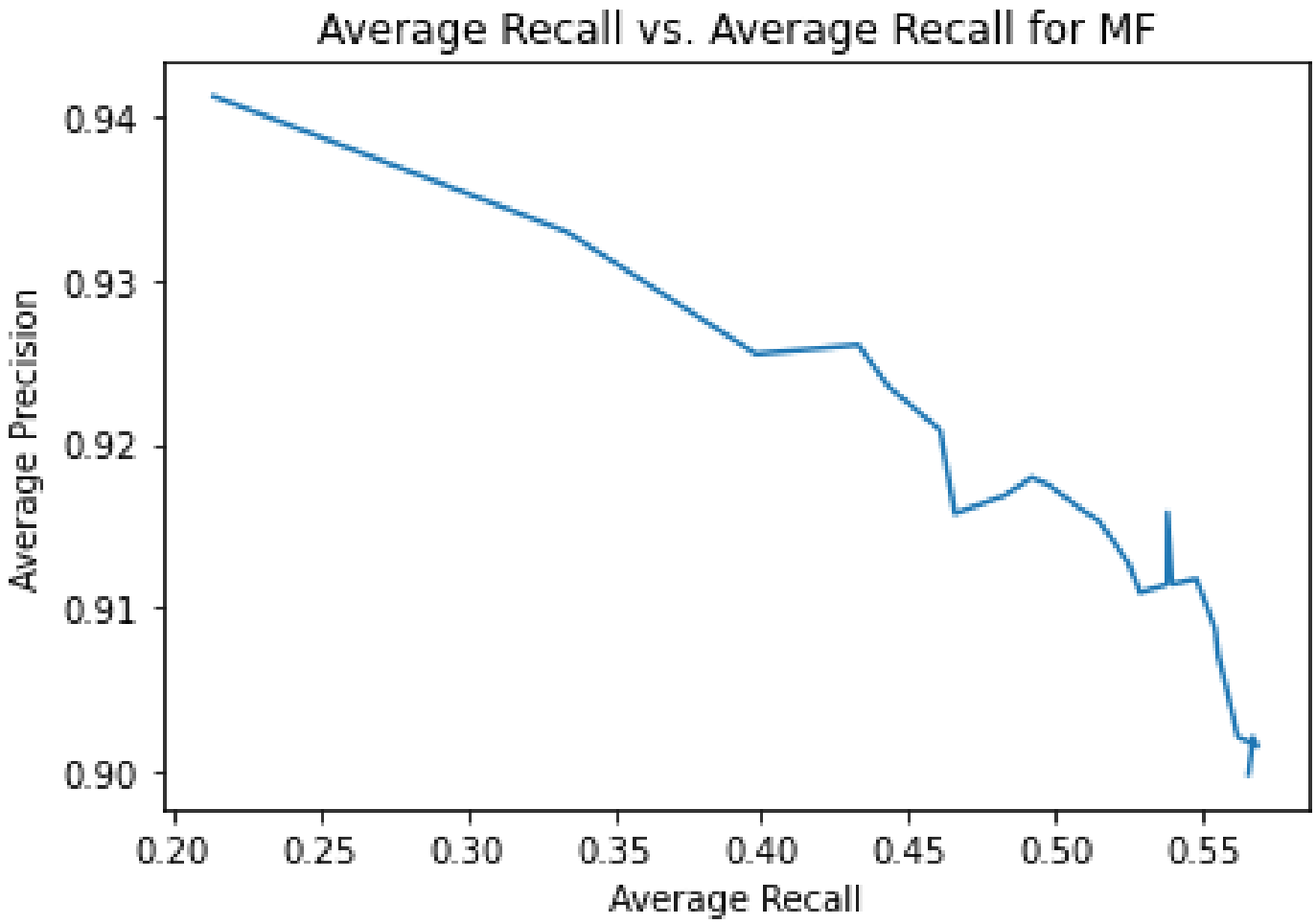


Figure 23: Average precision as function of average recall for MF filter predictions.

Question 39.

By comparing the precision vs recall curve for k-NN, NNMF and MF (figure 24), the recommendation list returned by MF with has the highest relevance while the recommendation list returned by NNMF has the lowest relevance. We determined this by seeing that MF has the higher precision and recall overall. Despite MF being better, it is only marginally so with performance being close between KNN, NNMF, and MF.

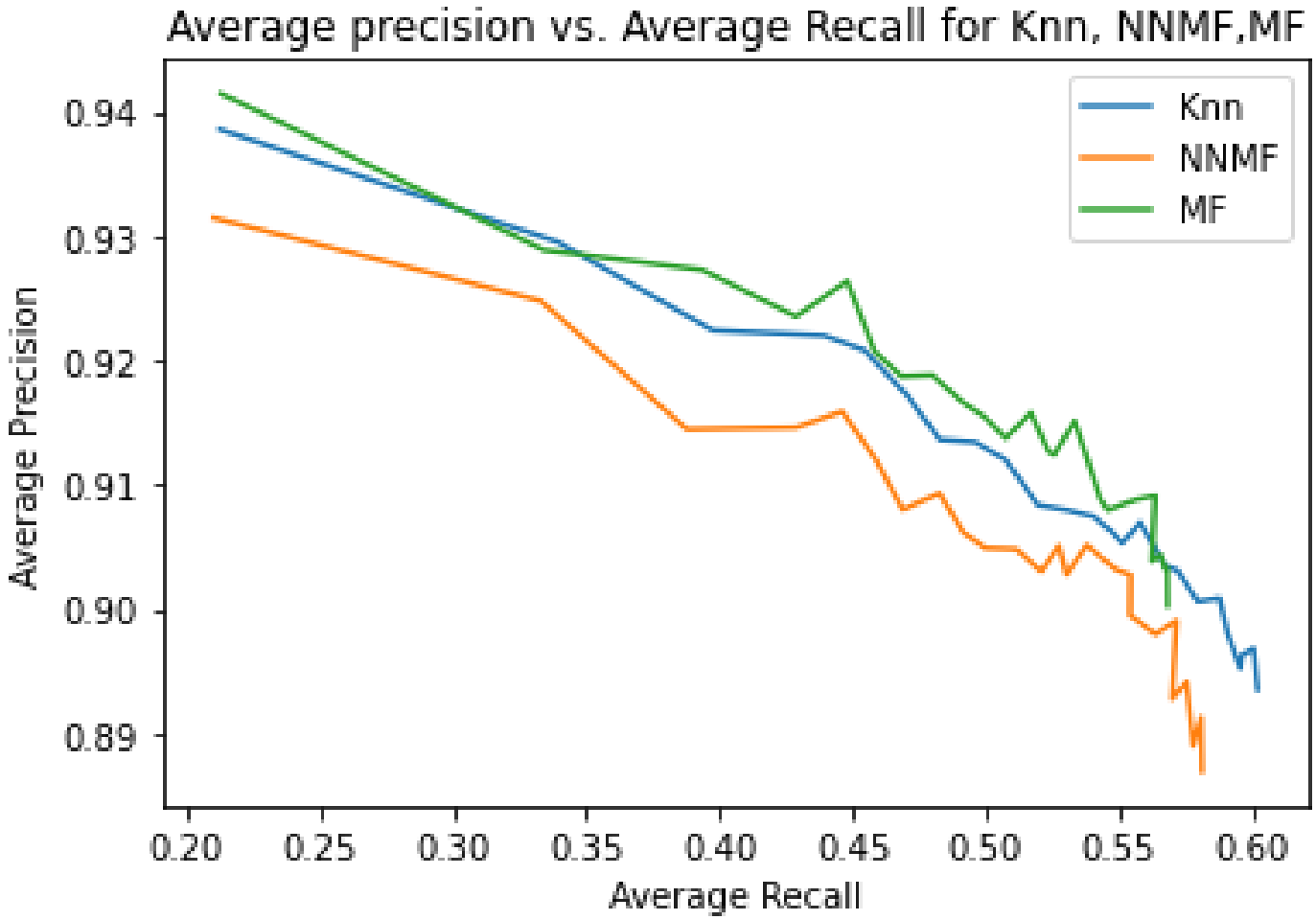


Figure 24: Average recall as function of t for MF filter predictions.

Appendix: Source Code

project3.py

```
import os
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
from pprint import pprint
# from textwrap import dedent
from itertools import product
import json


# from sklearn.metrics import confusion_matrix
# from scipy.optimize import linear_sum_assignment
from plotmat import plot_mat
# import umap

from helpers import get_movies_ratings, get_ratings_matrix,get_knn_performance_vs_k_over_k_folds,get_knn_performance_vs_k
    _over_k_folds_trimmed,get_knn_performance_fixed_k_different_thresholds,get_NNMF_performance_vs_k_over_10_folds
```

```
from helpers import get_MF_performance_vs_k_over_10_folds, get_MF_performance_vs_k_over_k_folds_trimmed,
    get_MF_performance_fixed_k_different_thresholds
from helpers import get_NNMF_performance_fixed_k_different_thresholds, get_NNMF_performance_vs_k_over_k_folds_trimmed,
    get_ratings_df,get_Naive_performance_10_folds_trimmed, q23_helper
from helpers import get_Naive_performance_10_folds, get_Naive_performance_10_folds_trimmed, q34_helper
from logger import logged
from sklearn.metrics import roc_curve, auc
from helpers import sorted_rating, sorted_groundtruth, num_intersection, user_precision_recall
from helpers import q36_helper, q37_helper, q38_helper, q39_helper
np.random.seed(0)
random.seed(0)

@logged
def q1():
  movies, ratings = get_movies_ratings()
  ratings_mat = get_ratings_matrix()
  # nMovies = len(set(movies.movieId))
  # nUsers = len(set(ratings.userId))
  nRatings = len(ratings)
  nPosRatings = np.prod(ratings_mat.shape)
  print(
  f"""
  Total possible ratings: {nPosRatings}
  Number Ratings: {nRatings}
  Sparsity: {nRatings/nPosRatings}
  """)

def q2():
    movies, ratings = get_movies_ratings()
    his = np.histogram(ratings.rating,bins=[0.01, 0.5001, 1.0001, 1.5001, 2.0001, 2.5001, 3.0001, 3.5001, 4.0001, 4.5001,
        5.0001])
    fig, ax = plt.subplots()
    plt.bar(his[1][1:],his[0],width=0.5,edgecolor='k')
    ax.set_xticks(his[1][1:])
    ax.set_xticklabels( ('0.5', '1', '1.5', '2', '2.5','3','3.5','4','4.5','5') )
    plt.xlabel("Movie Rating")
    plt.ylabel("Number of Movies with Rating")
    plt.title("Movie Ratings Histogram")
    plt.savefig(f"figures/q2_ratings_histogram.pdf")
    plt.show()

def q3():
  movies, ratings = get_movies_ratings()
  arrayMovieId = ratings.movieId
  u, count = np.unique(arrayMovieId, return_counts=True)
  sortedU = u[np.argsort(-count)]
  sortedCount = count[np.argsort(-count)]
  xIdx = np.arange(sortedU.shape[0])
  plt.title("Movie rating frequency")
  plt.xlabel("Movie Id ranking")
  plt.ylabel("Rating frequency")
  plt.plot(xIdx , sortedCount)
  # plt.show()
  plt.savefig(f"figures/q3_movie_ratings.pdf")

def q4():
  movies, ratings = get_movies_ratings()
  arrayUserId = ratings.userId
  u, count = np.unique(arrayUserId, return_counts=True)
  sortedU = u[np.argsort(-count)]
```

```
  sortedCount = count[np.argsort(-count)]
  xIndex = np.arange(sortedU.shape[0])
  plt.title("Distribution of ratings among users")
  plt.xlabel("User index ordered by decreasing frequency")
  plt.ylabel("Rating frequency")
  plt.plot(xIndex , sortedCount)
  # plt.show()
  plt.savefig(f"figures/q4_user_ratings.pdf")


def q6():
  movies, ratings = get_movies_ratings()
  arrayMovieId = ratings.movieId
  arrayRating = ratings.rating
  sortedM = arrayMovieId[np.argsort(arrayMovieId)]
  sortedR = arrayRating[np.argsort(arrayMovieId)]
  u, count = np.unique(sortedM, return_counts=True)
  index = 0
  ratingVar = np.zeros(u.shape)
  for i in range(len(u)):
    ratingVar[i] = np.var(sortedR[index:index+count[i]])
    index += count[i]
  print("Histogram with 0 variance")
  bins = np.arange(0, 6, 0.1)
  plt.hist(ratingVar, bins=bins)
  plt.title("Variance of the rating values received by each movie")
  plt.xlabel("Variance of the ratings")
  plt.ylabel("Number of movies")
  plt.show()
  plt.savefig(f"figures/q6_movie_rating_variance_histogram.pdf")
  plt.clf()
  print("Histogram without 0 variance")
  bins = np.arange(0.0001, 6, 0.1)
  plt.hist(ratingVar, bins=bins)
  plt.title("Variance of the rating values received by each movie without 0 variance")
  plt.xlabel("Variance of the ratings")
  plt.ylabel("Number of movies")
  plt.show()
  plt.savefig(f"figures/q6_movie_rating_variance_histogram_0_variance_removed.pdf")
  print("+ The number of variance 0 is",len(ratingVar)-np.count_nonzero(ratingVar))



def q10():
  kvals,rmse,mae = get_knn_performance_vs_k_over_k_folds()
  plt.plot(kvals , rmse,label="RMSE")
  plt.plot(kvals , mae,label="MAE")
  plt.xlabel("$K$ Nearest Neighbors")
  plt.ylabel("Accuracy (RMSE or MAE)")
  plt.title("Accuracy vs. $K$ in Movielens KNN")
  plt.legend()
  plt.savefig(f"figures/q10_error_vs_k_knn_kfold.pdf")



def q12_13_14():
  kvals,avg_rmses,avg_maes = get_knn_performance_vs_k_over_k_folds_trimmed()
  trimmers = list(avg_rmses.keys())
  nTrimmers = len(trimmers)
  fig,axs = plt.subplots(1,nTrimmers,figsize=(15,5),sharey=True)
```

```
  for i, (ax,trimmer) in enumerate(zip(axs,trimmers)):
    min_rmse = min(avg_rmses[trimmer])
    best_k_rmse = kvals[np.argmin(avg_rmses[trimmer])]
    min_mae = min(avg_maes[trimmer])
    best_k_mae = kvals[np.argmin(avg_maes[trimmer])]
    ax.plot(kvals,avg_rmses[trimmer],label=f"RMSE (minimum=${min_rmse :.3}$ at $k={best_k_rmse}$)")
    ax.plot(kvals,avg_maes[trimmer],label=f"MAE (minimum=${min_mae :.3}$ at $k={best_k_mae}$)")
    ax.set_title(f"Accuracy vs. $K$ in Movielens KNN\nWith Trimmer: {trimmer.__name__}")
    ax.set_xlabel("$K$ Nearest Neighbors")
    ax.legend()
    if i == 0:
      ax.set_ylabel("Accuracy (RMSE or MAE)")
  plt.tight_layout()
  plt.savefig(f"figures/q12_error_vs_k_knn_kfold_trimmed.pdf")

def plot_roc(targets, scores,ax,xlabel='False Positive Rate',ylabel='True Positive Rate',title="ROC Curve",nlabels=5):
  # title= f"ROC Curve with Threshold £{threshold}£"
  fprs,tprs,thresh = roc_curve(targets,scores)
  label_inds = [i for i in range(0,len(thresh),len(thresh)//nlabels)]
  if len(thresh)-1 not in label_inds:
    label_inds.pop()
    label_inds.append(len(thresh)-1)
  labelx = fprs[label_inds]
  labely = tprs[label_inds]
  labeltext = thresh[label_inds]
  roc_auc = auc(fprs,tprs)
  ax.plot(fprs, tprs, lw=2, label= f'AUC= {roc_auc : 0.04f}')
  ax.scatter(labelx,labely,marker="x")
  for x,y,lab in zip(labelx,labely,labeltext):
    ax.text(x+0.02,y-0.04,f"{lab:.02}",fontsize=12)
  ax.grid(color='0.7', linestyle='--', linewidth=1)
  # ax.set_xlim([-0.1, 1.1])
  # ax.set_ylim([0.0, 1.05])
  ax.set_xlabel(xlabel,fontsize=15)
  ax.set_ylabel(ylabel,fontsize=15)
  ax.set_title(title)
  ax.legend(loc="lower right",fontsize="x-small")
  for label in ax.get_xticklabels()+ax.get_yticklabels():
    label.set_fontsize(15)

def q15():
  best_k = 20 # value found in question 11, where the RMSE/MAE curves level out
  thresholds = (2.5,3,3.5,4)
  ground_truth_for_thresholds,predictions_for_thresholds = get_knn_performance_fixed_k_different_thresholds(best_k =
      best_k,thresholds=thresholds)
  fig,axs = plt.subplots(1,len(thresholds),figsize=(15,5),sharey=True)
  # i, (ax,threshold) = next(enumerate(zip(axs,thresholds)))
  for i, (ax,threshold) in enumerate(zip(axs,thresholds)):
    targets,scores = ground_truth_for_thresholds[threshold],predictions_for_thresholds[threshold]
    plot_roc(targets, scores,ax,title= f"ROC Curve with Threshold ${threshold}$")
  plt.savefig(f"figures/q15_thresholded_roc.pdf")

def q17():
  kvals,rmse,mae = get_NNMF_performance_vs_k_over_10_folds()
  plt.plot(kvals , rmse,label="RMSE")
  plt.plot(kvals , mae,label="MAE")
  print(rmse)
  print(mae)
  plt.xlabel("Number of Latent Factors")
  plt.ylabel("Accuracy (RMSE or MAE)")
```

```
  plt.title("Accuracy vs. Number of Latent Factors in Movielens NNMF")
  plt.legend()
  plt.savefig(f"figures/q17_error_vs_k_NNMF_kfold.pdf")

def q19_20_21():
  kvals,avg_rmses,avg_maes = get_NNMF_performance_vs_k_over_k_folds_trimmed()
  trimmers = list(avg_rmses.keys())
  nTrimmers = len(trimmers)
  fig,axs = plt.subplots(1,nTrimmers,figsize=(15,5),sharey=True)
  for i, (ax,trimmer) in enumerate(zip(axs,trimmers)):
    min_rmse = min(avg_rmses[trimmer])
    best_k_rmse = kvals[np.argmin(avg_rmses[trimmer])]
    min_mae = min(avg_maes[trimmer])
    best_k_mae = kvals[np.argmin(avg_maes[trimmer])]
    ax.plot(kvals,avg_rmses[trimmer],label=f"RMSE (minimum=${min_rmse :.3}$ at $k={best_k_rmse}$)")
    ax.plot(kvals,avg_maes[trimmer],label=f"MAE (minimum=${min_mae :.3}$ at $k={best_k_mae}$)")
    ax.set_title(f"Accuracy vs. Number of Latent Factors in Movielens NNMF\nWith Trimmer: {trimmer.__name__}")
    ax.set_xlabel("Number of Latent Factors")
    ax.legend()
    if i == 0:
      ax.set_ylabel("Accuracy (RMSE or MAE)")
  plt.tight_layout()
  plt.savefig(f"figures/q19_error_vs_k_knn_kfold_trimmed_1.pdf")

def q22():
  best_k = 20 # value found in question 11, where the RMSE/MAE curves level out
  thresholds = (2.5,3,3.5,4)
  ground_truth_for_thresholds,predictions_for_thresholds = get_NNMF_performance_fixed_k_different_thresholds(best_k =
      best_k,thresholds=thresholds)
  fig,axs = plt.subplots(1,len(thresholds),figsize=(15,5),sharey=True)
  # i, (ax,threshold) = next(enumerate(zip(axs,thresholds)))
  for i, (ax,threshold) in enumerate(zip(axs,thresholds)):
    targets,scores = ground_truth_for_thresholds[threshold],predictions_for_thresholds[threshold]
    # fprs,tprs,thresh = roc_curve(targets,scores)
    plot_roc(targets, scores,ax,title= f"ROC Curve with Threshold ${threshold}$")
  plt.savefig(f"figures/q22_thresholded_roc.pdf")

@logged
def q23():
  q23_helper()


def q24():
  kvals,rmse,mae = get_MF_performance_vs_k_over_10_folds()
  plt.plot(kvals , rmse,label="RMSE")
  plt.plot(kvals , mae,label="MAE")
  plt.xlabel("Number of Latent Factors")
  plt.ylabel("Accuracy (RMSE or MAE)")
  plt.title("Accuracy vs. Number of Latent Factors in Movielens MF")
  plt.legend()
  plt.savefig(f"figures/q24_error_vs_k_MF_kfold.pdf")

def q26_27_28():
  kvals,avg_rmses,avg_maes = get_MF_performance_vs_k_over_k_folds_trimmed()
  trimmers = list(avg_rmses.keys())
  nTrimmers = len(trimmers)
  fig,axs = plt.subplots(1,nTrimmers,figsize=(15,5),sharey=True)
  for i, (ax,trimmer) in enumerate(zip(axs,trimmers)):
    min_rmse = min(avg_rmses[trimmer])
    best_k_rmse = kvals[np.argmin(avg_rmses[trimmer])]
```

```
    min_mae = min(avg_maes[trimmer])
    best_k_mae = kvals[np.argmin(avg_maes[trimmer])]
    ax.plot(kvals,avg_rmses[trimmer],label=f"RMSE (minimum=${min_rmse :.3}$ at $k={best_k_rmse}$)")
    ax.plot(kvals,avg_maes[trimmer],label=f"MAE (minimum=${min_mae :.3}$ at $k={best_k_mae}$)")
    ax.set_title(f"Accuracy vs. $K$ in Movielens MF\nWith Trimmer: {trimmer.__name__}")
    ax.set_xlabel("$K$ Nearest Neighbors")
    ax.legend()
    if i == 0:
      ax.set_ylabel("Accuracy (RMSE or MAE)")
  plt.tight_layout()
  plt.savefig(f"figures/q26_error_vs_k_MF_kfold_trimmed.pdf")

def q29():
  best_k = 20 # value found in question 11, where the RMSE/MAE curves level out
  thresholds = (2.5,3,3.5,4)
  ground_truth_for_thresholds,predictions_for_thresholds = get_MF_performance_fixed_k_different_thresholds(best_k =
      best_k,thresholds=thresholds)
  fig,axs = plt.subplots(1,len(thresholds),figsize=(15,5),sharey=True)
  # i, (ax,threshold) = next(enumerate(zip(axs,thresholds)))
  for i, (ax,threshold) in enumerate(zip(axs,thresholds)):
    targets,scores = ground_truth_for_thresholds[threshold],predictions_for_thresholds[threshold]
    # fprs,tprs,thresh = roc_curve(targets,scores)
    plot_roc(targets, scores,ax,title= f"ROC Curve with Threshold ${threshold}$")
  plt.savefig(f"figures/q29_thresholded_roc.pdf")

def q30():
  rmse = get_Naive_performance_10_folds()
  print(rmse)

def q31_32_33():
  avg_rmses = get_Naive_performance_10_folds_trimmed()
  trimmers = list(avg_rmses.keys())
  for trimmer in trimmers:
    print(avg_rmses[trimmer])


def q34():
    p1,p2,p3,p4,g1,g2,g3,g4 = q34_helper()
    fig,axs = plt.subplots(1,4,figsize=(15,5),sharey=True)

    targets1,scores1 = g1,p1
    plot_roc(targets1, scores1,axs[0],title= f"KNN ROC Curve")
    targets2,scores2 = g2,p2
    plot_roc(targets2, scores2,axs[1],title= f"NNMF ROC Curve")
    targets3,scores3 = g3,p3
    plot_roc(targets3, scores3,axs[2],title= f"MF ROC Curve")
    targets4,scores4 = g4,p4
    plot_roc(targets4, scores4,axs[3],title= f"Basefilter ROC Curve")

    plt.savefig(f"figures/q34_thresholded_roc_all.pdf")



def q36():
  q36_helper()


def q37():
  q37_helper()
```

```
def q38():
  q38_helper()


def q39():
  q39_helper()
  pass



# In [4]: ratings_mat
# Out[4]:
# <610x9724 sparse matrix of type '<class 'numpy.float64'>'
#   with 100836 stored elements in Dictionary Of Keys format>
if __name__=="__main__":
  # q1()
  # q2()
  # q3()
  # q4()
  # q6()
  # q10()
  # q12_13_14()
  # q15()
  # q17()
  # q19_20_21()
  # q22()
  # q24()
  # q26_27_28()
  # q29()
  # q30()
  # q31_32_33()
  # q1()
  # q12_13_14()
  # q19_20_21()
  # q26_27_28()
  # q31_32_33()
  # q24()
  # q23()
  # q34()
  q37()
  pass
```

helpers.py

```
import os
import numpy as np
import pandas as pd
from scipy.sparse import dok_matrix
# https://gist.github.com/pankajti/e631e8f6ce067fc76dfacedd9e4923ca#file-surprise_knn_recommendation-ipynb
from surprise import SVD
from surprise import Dataset
from surprise import accuracy
from surprise.model_selection import train_test_split
from surprise import KNNBasic,  KNNWithMeans, KNNBaseline
from surprise.model_selection import KFold
from surprise.model_selection.split import ShuffleSplit
from surprise.trainset import Trainset
```

```
from surprise import Reader
from surprise import NormalPredictor
from surprise.model_selection import cross_validate
from surprise import similarities
# import seaborn as sns
from surprise.model_selection import GridSearchCV
from sklearn.metrics import roc_curve
from surprise import NMF
from surprise.prediction_algorithms.baseline_only import BaselineOnly
from caching import cached
from sklearn.decomposition import NMF as NMF_matrix
from surprise.model_selection import train_test_split
from collections import defaultdict
import matplotlib.pyplot as plt
# from sklearn.decomposition import NMF




def get_ml_filenames():
  mldir = "input/ml-latest-small"
  return [f"{mldir}/{fname}" for fname in os.listdir(mldir) if fname.endswith('.csv')]

# the 'movieId' column does not consist of consecutive integers starting at 0. This makes it so.
# the 'userId' column does not consist of consecutive integers starting at 0. This makes it so.
# a movieId and userId now corresponds to the appropriate column/row in the ratings matrix
def fix_indices(data):
  movies = data['movies'].copy()
  ratings = data['ratings'].copy()
  ratings.drop_duplicates(subset = 'movieId',inplace=True)
  movies = movies.merge(ratings,on='movieId',how='left',indicator=True)
  movies = movies.loc[movies['_merge'] == 'both']
  movies['new_index']= range(len(movies))
  for k,df in data.items():
    if 'movieId' in df:
      # df = df.loc[df.movieId.isin(movies.movieId)]
      df.drop(df[~df.movieId.isin(movies.movieId)].index,inplace=True)
      df['movieId'] = pd.merge(df,movies,how='left',on='movieId')['new_index'].values
    if 'userId' in df:
      df['userId'] -= 1
    data[k] = df
  return data

# returns {"links" : <links dataframe>, ..., "tags" : <tags dataframe>}
def get_ml_data():
  return fix_indices({ fname.split('.')[0].rsplit("/",1)[-1] : pd.read_csv(fname) for fname in get_ml_filenames()})

def get_movies_ratings():
  data = get_ml_data()
  return data['movies'],data['ratings']

def get_ratings_df():
  data = get_ml_data()
  return data['ratings']

#requires that all the movies considered are in the ratings df and have labels in 0,....,nMovies
def get_ratings_matrix(ratings=None):
  if ratings is None:
    ratings = get_ratings_df()
  nMovies = len(set(ratings.movieId))
```

```
  nUsers = len(set(ratings.userId))
  ratings_mat = dok_matrix((nUsers,nMovies))
  for userId, movieId, rating in ratings[['userId','movieId','rating']].values:
    ratings_mat[userId,movieId] = rating #subtract 1 because the indices start at 1
  return ratings_mat


def get_accuracies(predictions):
  return accuracy.rmse(predictions,verbose=False) , accuracy.mae(predictions,verbose=False)


@cached
def get_knn_performance_vs_k_over_k_folds():
  movies, ratings = get_movies_ratings()
  reader = Reader(rating_scale = (0.5,5))
  data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
  # anti_set = data.build_full_trainset().build_anti_testset()
  kvals = list(range(2,101,2))
  kf = KFold(n_splits=10)
  folds = list(kf.split(data))
  performance = np.array([
    np.mean([
      get_accuracies(KNNBasic(k=k_nn,sim_options={'name': 'pearson'},verbose=False).fit(trainset).test(testset)) for
          trainset,testset in folds
    ],axis=0)
    for k_nn in kvals])
  rmse,mae = performance[:,0],performance[:,1]
  return kvals,rmse,mae

@cached
def get_NNMF_performance_vs_k_over_10_folds():
  _, ratings = get_movies_ratings()
  reader = Reader(rating_scale = (0.5, 5))
  data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']],reader)
  kvals = list(range(2,52,2))
  kf = KFold(n_splits=10)
  folds = list(kf.split(data))
  performance = np.array([
    np.mean([
      get_accuracies(NMF(n_factors=k).fit(trainset).test(testset)) for trainset,testset in folds
    ],axis=0)
    for k in kvals])
  rmse,mae = performance[:,0],performance[:,1]
  return kvals,rmse,mae

@cached
def get_MF_performance_vs_k_over_10_folds():
  _, ratings = get_movies_ratings()
  reader = Reader(rating_scale = (0.5, 5))
  data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']],reader)
  kvals = list(range(2,52,2))
  kf = KFold(n_splits=10)
  folds = list(kf.split(data))
  performance = np.array([
    np.mean([
      get_accuracies(SVD(n_factors=k).fit(trainset).test(testset)) for trainset,testset in folds
    ],axis=0)
    for k in kvals])
  rmse,mae = performance[:,0],performance[:,1]
  return kvals,rmse,mae
```

```
def apply_threshold(value,threshold):
  return 0 if value < threshold else 1


@cached
def get_knn_performance_fixed_k_different_thresholds(best_k=20,thresholds=(2.5,3,3.5,4)):
  ratings_src = get_ratings_df()
  reader = Reader(rating_scale = (0.5,5))
  ground_truth_for_thresholds = {threshold : None for threshold in thresholds}
  predictions_for_thresholds = {threshold : None for threshold in thresholds}
  # threshold=thresholds[0]
  for threshold in thresholds:
    ratings = ratings_src.copy()
    # ratings.rating = (ratings.rating > threshold).astype(int)
    data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
    shuf = ShuffleSplit(n_splits=1,test_size=0.1,random_state=0)
    trainset, testset = next(shuf.split(data))
    classifier = KNNBasic(k=best_k,sim_options={'name': 'pearson'},verbose=False).fit(trainset)
    predictions = classifier.test(testset)
    predictions_for_thresholds[threshold]  = [prediction.est for prediction in predictions]
    ground_truth_for_thresholds[threshold] = [apply_threshold(prediction.r_ui,threshold) for prediction in predictions]
  return ground_truth_for_thresholds, predictions_for_thresholds

@cached
def get_NNMF_performance_fixed_k_different_thresholds(best_k=20,thresholds=(2.5,3,3.5,4)):
  ratings_src = get_ratings_df()
  reader = Reader(rating_scale = (0.5,5))
  ground_truth_for_thresholds = {threshold : None for threshold in thresholds}
  predictions_for_thresholds = {threshold : None for threshold in thresholds}
  for threshold in thresholds:
    ratings = ratings_src.copy()
    # ratings.rating = (ratings.rating > threshold).astype(int)
    data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
    shuf = ShuffleSplit(n_splits=1,test_size=0.1,random_state=0)
    trainset, testset = next(shuf.split(data))
    classifier = NMF(n_factors=best_k).fit(trainset)
    predictions = classifier.test(testset)
    predictions_for_thresholds[threshold]  = [prediction.est for prediction in predictions]
    ground_truth_for_thresholds[threshold] = [apply_threshold(prediction.r_ui,threshold) for prediction in predictions]
  return ground_truth_for_thresholds, predictions_for_thresholds


def apply_threshold_to_values(values, threshold):
  return [apply_threshold(value,threshold) for value in values]


@cached
def get_MF_performance_fixed_k_different_thresholds(best_k=20,thresholds=(2.5,3,3.5,4)):
  ratings_src = get_ratings_df()
  reader = Reader(rating_scale = (0.5,5))
  ground_truth_for_thresholds = {threshold : None for threshold in thresholds}
  predictions_for_thresholds = {threshold : None for threshold in thresholds}
  for threshold in thresholds:
    ratings = ratings_src.copy()
    # ratings.rating = (ratings.rating > threshold).astype(int)
    data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
    shuf = ShuffleSplit(n_splits=1,test_size=0.1,random_state=0)
```

```
    trainset, testset = next(shuf.split(data))
    classifier = SVD(n_factors=best_k).fit(trainset)
    predictions = classifier.test(testset)
    predictions_for_thresholds[threshold]  = [prediction.est for prediction in predictions]
    ground_truth_for_thresholds[threshold] = [apply_threshold(prediction.r_ui,threshold) for prediction in predictions]
  return ground_truth_for_thresholds, predictions_for_thresholds


@cached
def get_knn_performance_vs_k_over_k_folds_trimmed():
  trimmers = (get_popular_movieIds, get_unpopular_movieIds, get_high_variance_movieIds)
  ratings = get_ratings_df()
  reader = Reader(rating_scale = (0.5,5))
  data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
  # anti_set = data.build_full_trainset().build_anti_testset()
  kvals = list(range(2,101,2))
  kf = KFold(n_splits=10)
  folds = list(kf.split(data))
  trimmed_folds = [ (trainset, {trimmer: trim_testset(trimmer,testset) for trimmer in trimmers}) for (trainset,testset)
      in folds ]
  print("Got trimmed folds.")
  avg_rmses = {trimmer : [] for trimmer in trimmers}
  avg_maes = {trimmer : [] for trimmer in trimmers}
  for k_nn in kvals:
    print(f"k={k_nn}")
    rmses = {trimmer : [] for trimmer in trimmers}
    maes = {trimmer : [] for trimmer in trimmers}
    for trainset,trimmed_testsets in trimmed_folds:
      classifier = KNNBasic(k=k_nn,sim_options={'name': 'pearson'},verbose=False).fit(trainset)
      for trimmer,testset in trimmed_testsets.items():
        rmse, mae = get_accuracies(classifier.test(testset))
        rmses[trimmer].append(rmse)
        maes[trimmer].append(mae)
    for trimmer in trimmers:
      avg_rmses[trimmer].append(np.mean(rmses[trimmer]))
      avg_maes[trimmer].append(np.mean(maes[trimmer]))
  return kvals,avg_rmses,avg_maes

@cached
def get_NNMF_performance_vs_k_over_k_folds_trimmed():
  trimmers = (get_popular_movieIds, get_unpopular_movieIds, get_high_variance_movieIds)
  ratings = get_ratings_df()
  reader = Reader(rating_scale = (0.5,5))
  data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
  kvals = list(range(2,52,2))
  kf = KFold(n_splits=10)
  folds = list(kf.split(data))
  trimmed_folds = [ (trainset, {trimmer: trim_testset(trimmer,testset) for trimmer in trimmers}) for (trainset,testset)
      in folds ]
  print("Got trimmed folds.")
  avg_rmses = {trimmer : [] for trimmer in trimmers}
  avg_maes = {trimmer : [] for trimmer in trimmers}
  for k in kvals:
    print(f"k={k}")
    rmses = {trimmer : [] for trimmer in trimmers}
    maes = {trimmer : [] for trimmer in trimmers}
    for trainset,trimmed_testsets in trimmed_folds:
      classifier = NMF(n_factors=k).fit(trainset)
      for trimmer, testset in trimmed_testsets.items():
        rmse, mae = get_accuracies(classifier.test(testset))
```

```
        rmses[trimmer].append(rmse)
        maes[trimmer].append(mae)
    for trimmer in trimmers:
      avg_rmses[trimmer].append(np.mean(rmses[trimmer]))
      avg_maes[trimmer].append(np.mean(maes[trimmer]))
  return kvals,avg_rmses,avg_maes

@cached
def get_MF_performance_vs_k_over_k_folds_trimmed():
  trimmers = (get_popular_movieIds, get_unpopular_movieIds, get_high_variance_movieIds)
  ratings = get_ratings_df()
  reader = Reader(rating_scale = (0.5,5))
  data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
  kvals = list(range(2,52,2))
  kf = KFold(n_splits=10)
  folds = list(kf.split(data))
  trimmed_folds = [ (trainset, {trimmer: trim_testset(trimmer,testset) for trimmer in trimmers}) for (trainset,testset)
      in folds ]
  print("Got trimmed folds.")
  avg_rmses = {trimmer : [] for trimmer in trimmers}
  avg_maes = {trimmer : [] for trimmer in trimmers}
  for k in kvals:
    print(f"k={k}")
    rmses = {trimmer : [] for trimmer in trimmers}
    maes = {trimmer : [] for trimmer in trimmers}
    for trainset,trimmed_testsets in trimmed_folds:
      classifier = SVD(n_factors=k).fit(trainset)
      for trimmer, testset in trimmed_testsets.items():
        rmse, mae = get_accuracies(classifier.test(testset))
        rmses[trimmer].append(rmse)
        maes[trimmer].append(mae)
    for trimmer in trimmers:
      avg_rmses[trimmer].append(np.mean(rmses[trimmer]))
      avg_maes[trimmer].append(np.mean(maes[trimmer]))
  return kvals,avg_rmses,avg_maes


def trim_testset(trimmer,testset):
  desired_ids = trimmer()
  return [tup for tup in testset if tup[1] in desired_ids]

def get_popular_movieIds():
  ratings_mat = get_ratings_matrix()
  return (np.sum(ratings_mat,axis=0) > 2).nonzero()[1]

def get_unpopular_movieIds():
  ratings_mat = get_ratings_matrix()
  return (np.sum(ratings_mat,axis=0) <= 2).nonzero()[1]

def get_high_variance_movieIds():
  ratings_mat = get_ratings_matrix()
  at_least_5_ratings = (np.sum(ratings_mat,axis=0) >= 5).nonzero()[1]
  # at_least_one_5 = (ratings_mat.tocsc().max(axis=0).toarray()==5).nonzero()[1].tolist()
  # col_means = np.sum(ratings_mat,axis=0)/(np.sum(ratings_mat!=0,axis=0))
  col_variances = np.array([np.var(list(ratings_mat[:,j].values())) for j in range(ratings_mat.shape[1])])
  high_variance = (col_variances >= 2).nonzero()[0].tolist()
  return sorted(list(set(at_least_5_ratings).intersection(set(high_variance))))


def get_Naive_performance_10_folds():
```

```
    _, ratings = get_movies_ratings()
    reader = Reader(rating_scale = (0.5, 5))
    data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']],reader)
    kf = KFold(n_splits=10)
    folds = list(kf.split(data))
    bsl_options = {'reg_u': 0, 'reg_i': 0}
    performance = np.array([
        np.mean([
            accuracy.rmse(BaselineOnly(bsl_options=bsl_options).fit(trainset).test(testset)) for trainset,testset in
                folds])])
    rmse = performance
    return rmse

def get_Naive_performance_10_folds_trimmed():
    trimmers = (get_popular_movieIds, get_unpopular_movieIds, get_high_variance_movieIds)
    ratings = get_ratings_df()
    reader = Reader(rating_scale = (0.5,5))
    data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
    kf = KFold(n_splits=10)
    folds = list(kf.split(data))
    trimmed_folds = [ (trainset, {trimmer: trim_testset(trimmer,testset) for trimmer in trimmers}) for (trainset,testset)
        in folds ]
    print("Got trimmed folds.")
    bsl_options = {'reg_u': 0, 'reg_i': 0}
    avg_rmses = {trimmer : [] for trimmer in trimmers}

    rmses = {trimmer : [] for trimmer in trimmers}
    for trainset,trimmed_testsets in trimmed_folds:
        classifier = BaselineOnly(bsl_options=bsl_options).fit(trainset)
        for trimmer, testset in trimmed_testsets.items():
            rmse = accuracy.rmse(classifier.test(testset))
            rmses[trimmer].append(rmse)
    for trimmer in trimmers:
        avg_rmses[trimmer].append(np.mean(rmses[trimmer]))
    return avg_rmses


def q23_helper():
  reader = Reader(rating_scale = (0.5, 5))
  ratings = get_ratings_df()
  data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
  train_data = data.build_full_trainset()
  model = NMF(n_factors=20)
  model.fit(train_data)
  V = model.qi
  movies_df = pd.read_csv('input/ml-latest-small/movies.csv', names=['movieid','title','genres'])
  for i in [0,2,4,6,12,14,16,18]:
    print(f"Column {i}")
    top_10 = np.argsort(V[:,i])[-11:-1]
    for j in top_10:
      print(movies_df['genres'][j])


# def shuffled_inplace(arr):
#   np.random.shuffle(arr)
#   return arr

# def get_knn_predictions(k=5):
#   movies, ratings = get_movies_ratings()
#   ratings_mat = get_ratings_matrix()
```

```
#   nUsers, nMovies = ratings_mat.shape
#   pearson = get_pearson_matrix(ratings_mat)
#   mu = np.sum(ratings_mat,axis=1)/(np.sum(ratings_mat!=0,axis=1))
#   row_threshold = np.sort(pearson,axis=1)[:,-k]
#   knn_mask = np.array([[el >= row_threshold[i] for el in row] for i,row in enumerate(pearson)]) #note - this may have
    MORE THAN k nearest neighbors if there are ties, but that shouldn't affect the prediction function AT ALL.
#   for row in knn_mask:
#     row[shuffled_inplace(row.nonzero()[0])[k:]]=False #zero out all but k nearest neighbors

#   # knn_mask = pearson >= row_threshold
#   pearson_knn_only = np.multiply(pearson,knn_mask)

  # predictions = mu +
# def get_pearson_matrix(ratings_mat=None):
#   if ratings_mat is None:
#     ratings_mat = get_ratings_matrix()
#   mu = np.sum(ratings_mat,axis=1)/(np.sum(ratings_mat!=0,axis=1))
#   centered = (ratings_mat !=0).multiply(ratings_mat - mu ).tocsc() #subtract mean of each row's nonzero entries, store
    as csc for matrix multiplication
#   scales = np.sqrt((centered !=0) @ centered.T.power(2))
#   scales = scales.multiply(scales.T)
#   scales.data = np.reciprocal(scales.data)
#   pearson = ((centered @ centered.T).multiply( scales  )).toarray()
#   return pearson

# This function is extremely slow. It checks that the pearson
# matrix from get_pearson_matrix is correct. Calculating with
# Python loops is too slow and the matrix operations in
# get_pearson_matrix (which are executed by Numpy in C) should
# be used.
# def check_pearson_matrix(tol=0.05):
#   movies, ratings = get_movies_ratings()
#   ratings_mat = get_ratings_matrix()
#   pearson_fast = get_pearson_matrix(ratings_mat)
#   print("Got fast pearson matrix.")
#   nUsers, nMovies = ratings_mat.shape
#   userMeans = np.sum(ratings_mat,axis=1)/(np.sum(ratings_mat!=0,axis=1))
#   pearson = np.zeros((nUsers,nUsers))
#   for u in range(nUsers):
#     if u %10 == 0:
#       print(f"{u=}")
#     uvSums = np.zeros(nUsers)
#     uSums = np.zeros(nUsers)
#     vSums = np.zeros(nUsers)
#     for (_,k), ruk in ratings_mat[u,:].items():
#       for (v,_), rvk in ratings_mat[:,k].items():
#         uvSums[v] += (ruk - userMeans[u])*(rvk-userMeans[v])
#         uSums[v] += (ruk-userMeans[u])**2
#         vSums[v] += (rvk-userMeans[v])**2
#     pearson[u,:] = uvSums / np.multiply(np.sqrt(uSums),np.sqrt(vSums))
#     if any(abs(pearson[u,:] - pearson_fast[u,:]) > tol):
#       wrong = np.array(abs(pearson[u,:] - pearson_fast[u,:]) > tol).nonzero()[0].tolist()
#       print(f"{len(wrong)} elements of pearson matrix in row {u} are incorrect: {wrong}")
#       for ind in wrong:
#         print(f"pearson fast : {pearson_fast[u,ind]}  {pearson[u,ind]} : pearson")
#   return pearson

##
## FROM ranking.py
##
```

```
#return a list contain sorted rating and corresponding
#movie ID(incluidng movie without groundtruth rating) for each User
def sorted_rating():
    kf = KFold(n_splits=10)
    algo = KNNBasic(sim_options={'name': 'pearson'},verbose=False)
    movies, ratings = get_movies_ratings()
    reader = Reader(rating_scale = (0.5,5))
    data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
    total_prediction=[]
    for trainset, testset in kf.split(data):
        # train and test algorithm.
        algo.fit(trainset)
        predictions = algo.test(testset)
        total_prediction.append(predictions)
    total_prediction=pd.DataFrame(np.concatenate(total_prediction, axis=0))
    #set of those user and item pairs for which a rating doesn't exist in original dataset.
    anti_set = data.build_full_trainset().build_anti_testset()
    q=algo.test(anti_set)
    t=np.asarray(total_prediction)
    new=np.concatenate((t,q))
    movies, ratings = get_movies_ratings()
    arrayUserID = new[:,0]
    arrayMovieID = new[:,1]
    arrayRating = new[:,3]
    sortedU = arrayUserID[np.argsort(arrayUserID)]
    sortedM = arrayMovieID[np.argsort(arrayUserID)]
    sortedR = arrayRating[np.argsort(arrayUserID)]
    u, count = np.unique(sortedU, return_counts=True)
    index = 0
    user_rating = []
    user_movie = []
    for i in range(len(u)):
        user_rating.append(sortedR[index:index+count[i]])
        index += count[i]
    index = 0
    for i in range(len(u)):
        user_movie.append(sortedM[index:index+count[i]])
        index += count[i]
    sorted_user_rating=[]
    sorted_user_movie=[]
    for i in range(len(user_rating)):
        sorted_user_rating.append(user_rating[i][np.argsort(-user_rating[i])])
        sorted_user_movie.append(user_movie[i][np.argsort(-user_rating[i])])
    return sorted_user_rating, sorted_user_movie

#return a list contain sorted rating and corresponding
#movie ID(with groundtruth rating) for each User
def sorted_groundtruth():
  movies, ratings = get_movies_ratings()
  ratings[['userId','movieId','rating']]
  arrayUserID = np.asarray(ratings[['userId']])[:,0]
  arrayMovieID =  np.asarray(ratings[['movieId']])[:,0]
  arrayRating =  np.asarray(ratings[['rating']])[:,0]
  sortedU = arrayUserID[np.argsort(arrayUserID)]
  sortedM = arrayMovieID[np.argsort(arrayUserID)]
  sortedR = arrayRating[np.argsort(arrayUserID)]
  u, count = np.unique(sortedU, return_counts=True)
  index = 0
  user_rating = []
  user_movie = []
```

```
  for i in range(len(u)):
    user_rating.append(sortedR[index:index+count[i]])
    index += count[i]
  index = 0
  for i in range(len(u)):
    user_movie.append(sortedM[index:index+count[i]])
    index += count[i]
  truth_sorted_user_rating=[]
  truth_sorted_user_movie=[]
  for i in range(len(user_rating)):
    u=user_rating[i][np.argsort(-user_rating[i])]
    t=user_movie[i][np.argsort(-user_rating[i])]
    truth_sorted_user_rating.append(u[u>3])
    truth_sorted_user_movie.append(t[u>3])
  return truth_sorted_user_rating, truth_sorted_user_movie

def num_intersection(lst1, lst2):
    return len(list(set(lst1) & set(lst2)))

# def user_precision_recall():
#     sorted_user_rating, sorted_user_movie = sorted_rating()
#     truth_sorted_user_rating, truth__sorted_user_movie = sorted_groundtruth()
#     empties=[]
#     for s in truth__sorted_user_movie:
#         if len(s) == 0:
#             empties.append(truth__sorted_user_movie.index(s))
#     truth__sorted_user_movie.pop(empties[0])
#     sorted_user_movie.pop(empties[0])
#     precision=[]
#     recall=[]
#     t_list=[]
#     t=20
#     for i in range(len (truth__sorted_user_movie)):
#         tru=truth__sorted_user_movie[i]
#         s=sorted_user_movie[i][:t]
#         num=num_intersection(tru,s)
#         precision.append(num/t)
#         recall.append(num/len(truth__sorted_user_movie[i]))
#     return precision, recall


def user_precision_recall(predictions, t_value):
  record = defaultdict(list)
  for uid, _, r_actual, r_est, _ in predictions:
    record[uid].append((r_actual, r_est))

  precision_list = []
  recall_list = []

  for uid, ratings in record.items():
    ratings.sort(key=lambda x: x[1], reverse=True)
    G_size = 0
    S_t_size = 0
    G_and_S_t = 0
    for i, _ in ratings:
      if i >= 3:
        G_size += 1
    if G_size != 0:
      if len(ratings) >= t_value:
        for i, j in ratings[:t_value]:
```

```
            if j >= 3:
              S_t_size += 1
            if i >= 3 and j >=3:
              G_and_S_t += 1
          if S_t_size != 0:
            precision_list.append(G_and_S_t / S_t_size)
          else:
            precision_list.append(1)
          recall_list.append(G_and_S_t / G_size)
    return sum(precision_list)/len(precision_list), sum(recall_list)/len(recall_list)



def q36_helper():
    _, ratings = get_movies_ratings()
    reader = Reader(rating_scale = (0.5,5))
    data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
    kf = KFold(n_splits=10)
    precision = []
    recall = []
    for t_value in range(1, 26):
      precision_t = []
      recall_t = []
      for trainset, testset in kf.split(data):
        model = KNNBasic(k=20,sim_options={'name': 'pearson'},verbose=False)
        model.fit(trainset)
        predictions = model.test(testset)
        user_precision_recall(predictions, t_value)
        precision_iter, recall_iter = user_precision_recall(predictions, t_value)
        precision_t.append(precision_iter)
        recall_t.append(recall_iter)
      precision.append(sum(precision_t)/len(precision_t))
      recall.append(sum(recall_t)/len(recall_t))

    plt.plot(list(range(1, 26)), precision)
    plt.xlabel('t')
    plt.ylabel('Average Precision')
    plt.title('Average Precision vs. T for KNN')
    plt.show()
    plt.savefig(f"figures/q36_precision.png")

    plt.plot(list(range(1, 26)), recall)
    plt.xlabel('t')
    plt.ylabel('Average Recall')
    plt.title('Average Recall vs. T for KNN')
    plt.show()
    plt.savefig(f"figures/q36_recall.png")

    plt.plot(recall, precision)
    plt.xlabel('Average Recall')
    plt.ylabel('Average Precision')
    plt.title('Average Recall vs. Average Recall for KNN')
    plt.show()
    plt.savefig(f"figures/q36_precision_recall.png")

def q37_helper():
    _, ratings = get_movies_ratings()
    reader = Reader(rating_scale = (0.5,5))
    data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
```

```
  kf = KFold(n_splits=10)
  precision = []
  recall = []
  for t_value in range(1, 26):
    precision_t = []
    recall_t = []
    for trainset, testset in kf.split(data):
      model = NMF(n_factors=20)
      model.fit(trainset)
      predictions = model.test(testset)
      user_precision_recall(predictions, t_value)
      precision_iter, recall_iter = user_precision_recall(predictions, t_value)
      precision_t.append(precision_iter)
      recall_t.append(recall_iter)
    precision.append(sum(precision_t)/len(precision_t))
    recall.append(sum(recall_t)/len(recall_t))

  plt.plot(list(range(1, 26)), precision)
  plt.xlabel('t')
  plt.ylabel('Average Precision')
  plt.title('Average Precision vs. T for NMF')
  plt.show()
  plt.savefig(f"figures/q37_precision.png")

  plt.plot(list(range(1, 26)), recall)
  plt.xlabel('t')
  plt.ylabel('Average Recall')
  plt.title('Average Recall vs. T for NMF')
  plt.show()
  plt.savefig(f"figures/q37_recall.png")

  plt.plot(recall, precision)
  plt.xlabel('Average Recall')
  plt.ylabel('Average Precision')
  plt.title('Average Recall vs. Average Recall for NMF')
  plt.show()
  plt.savefig(f"figures/q37_precision_recall.png")

def q38_helper():
  _, ratings = get_movies_ratings()
  reader = Reader(rating_scale = (0.5,5))
  data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
  kf = KFold(n_splits=10)
  precision = []
  recall = []
  for t_value in range(1, 26):
    precision_t = []
    recall_t = []
    for trainset, testset in kf.split(data):
      model = SVD(n_factors=20)
      model.fit(trainset)
      predictions = model.test(testset)
      user_precision_recall(predictions, t_value)
      precision_iter, recall_iter = user_precision_recall(predictions, t_value)
      precision_t.append(precision_iter)
      recall_t.append(recall_iter)
    precision.append(sum(precision_t)/len(precision_t))
    recall.append(sum(recall_t)/len(recall_t))

  plt.plot(list(range(1, 26)), precision)
```

```
  plt.xlabel('t')
  plt.ylabel('Average Precision')
  plt.title('Average Precision vs. T for MF')
  plt.show()
  plt.savefig(f"figures/q38_precision.png")

  plt.plot(list(range(1, 26)), recall)
  plt.xlabel('t')
  plt.ylabel('Average Recall')
  plt.title('Average Recall vs. T for MF')
  plt.show()
  plt.savefig(f"figures/q38_recall.png")

  plt.plot(recall, precision)
  plt.xlabel('Average Recall')
  plt.ylabel('Average Precision')
  plt.title('Average Recall vs. Average Recall for MF')
  plt.show()
  plt.savefig(f"figures/q38_precision_recall.png")

def q39_helper():
  _, ratings = get_movies_ratings()
  reader = Reader(rating_scale = (0.5,5))
  data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
  kf = KFold(n_splits=10)
  precision_Knn = []
  recall_Knn = []
  precision_NNMF = []
  recall_NNMF = []
  precision_MF= []
  recall_MF = []
  for t_value in range(1, 26):
    precision_t_Knn = []
    recall_t_Knn= []
    precision_t_NNMF = []
    recall_t_NNMF = []
    precision_t_MF = []
    recall_t_MF = []
    for trainset, testset in kf.split(data):
      MF = SVD(n_factors=20)
      NNMF = NMF(n_factors=20)
      Knn  = KNNBasic(k=20,sim_options={'name': 'pearson'},verbose=False)
      MF.fit(trainset)
      NNMF.fit(trainset)
      Knn.fit(trainset)
      predictions_Knn = Knn.test(testset)
      predictions_MF = MF.test(testset)
      predictions_NNMF = NNMF.test(testset)
      precision_iter_Knn, recall_iter_Knn = user_precision_recall(predictions_Knn, t_value)
      precision_iter_NNMF, recall_iter_NNMF = user_precision_recall(predictions_NNMF, t_value)
      precision_iter_MF, recall_iter_MF = user_precision_recall(predictions_MF, t_value)
      precision_t_Knn.append(precision_iter_Knn)
      recall_t_Knn.append(recall_iter_Knn)
      precision_t_NNMF.append(precision_iter_NNMF)
      recall_t_NNMF.append(recall_iter_NNMF)
      precision_t_MF.append(precision_iter_MF)
      recall_t_MF.append(recall_iter_MF)
    precision_Knn.append(sum(precision_t_Knn)/len(precision_t_Knn))
    recall_Knn.append(sum(recall_t_Knn)/len(recall_t_Knn))
    precision_NNMF.append(sum(precision_t_NNMF)/len(precision_t_NNMF))
```

```
    recall_NNMF.append(sum(recall_t_NNMF)/len(recall_t_NNMF))
    precision_MF.append(sum(precision_t_MF)/len(precision_t_MF))
    recall_MF.append(sum(recall_t_MF)/len(recall_t_MF))


  plt.plot(recall_Knn, precision_Knn,label='Knn')
  plt.plot(recall_NNMF, precision_NNMF,label='NNMF')
  plt.plot(recall_MF, precision_MF,label='MF')
  plt.xlabel('Average Recall')
  plt.ylabel('Average Precision')
  plt.title('Average precision vs. Average Recall for Knn, NNMF,MF')
  plt.legend()
  plt.show()
  plt.savefig(f"figures/q39.png")


def q34_helper():

    best_k = 20 # value found in question 11, where the RMSE/MAE curves level out
    thresholds = 3
    threshold = 3
    ratings_src = get_ratings_df()
    reader = Reader(rating_scale = (0.5,5))
    ground_truth_for_thresholds1 = {thresholds}
    predictions_for_thresholds1 = {thresholds}
    ratings = ratings_src.copy()
    data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
    shuf = ShuffleSplit(n_splits=1,test_size=0.1,random_state=0)
    trainset, testset = next(shuf.split(data))
    classifier = KNNBasic(k=best_k,sim_options={'name': 'pearson'},verbose=False).fit(trainset)
    predictions = classifier.test(testset)
    predictions_for_thresholds1  = [prediction.est for prediction in predictions]
    ground_truth_for_thresholds1 = [apply_threshold(prediction.r_ui,threshold) for prediction in predictions]

    ratings_src = get_ratings_df()
    reader = Reader(rating_scale = (0.5,5))
    ground_truth_for_thresholds2 = {thresholds}
    predictions_for_thresholds2 = {thresholds}
    ratings = ratings_src.copy()
    # ratings.rating = (ratings.rating > threshold).astype(int)
    data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
    shuf = ShuffleSplit(n_splits=1,test_size=0.1,random_state=0)
    trainset, testset = next(shuf.split(data))
    classifier = NMF(n_factors=best_k).fit(trainset)
    predictions = classifier.test(testset)
    predictions_for_thresholds2  = [prediction.est for prediction in predictions]
    ground_truth_for_thresholds2 = [apply_threshold(prediction.r_ui,threshold) for prediction in predictions]

    ratings_src = get_ratings_df()
    reader = Reader(rating_scale = (0.5,5))
    ground_truth_for_thresholds3 = {thresholds}
    predictions_for_thresholds3 = {thresholds}
    ratings = ratings_src.copy()
    # ratings.rating = (ratings.rating > threshold).astype(int)
    data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
    shuf = ShuffleSplit(n_splits=1,test_size=0.1,random_state=0)
    trainset, testset = next(shuf.split(data))
    classifier = SVD(n_factors=best_k).fit(trainset)
    predictions = classifier.test(testset)
    predictions_for_thresholds3 = [prediction.est for prediction in predictions]
```

```
    ground_truth_for_thresholds3 = [apply_threshold(prediction.r_ui,threshold) for prediction in predictions]

    ratings_src = get_ratings_df()
    reader = Reader(rating_scale = (0.5,5))
    ground_truth_for_thresholds4 = {thresholds}
    predictions_for_thresholds4 = {thresholds}
    bsl_options = {'reg_u': 0, 'reg_i': 0}
    ratings = ratings_src.copy()
    # ratings.rating = (ratings.rating > threshold).astype(int)
    data = Dataset.load_from_df(ratings[['userId','movieId','rating']],reader)
    shuf = ShuffleSplit(n_splits=1,test_size=0.1,random_state=0)
    trainset, testset = next(shuf.split(data))
    classifier = BaselineOnly(bsl_options=bsl_options).fit(trainset)
    predictions = classifier.test(testset)
    predictions_for_thresholds4 = [prediction.est for prediction in predictions]
    ground_truth_for_thresholds4 = [apply_threshold(prediction.r_ui,threshold) for prediction in predictions]

    return
        predictions_for_thresholds1,predictions_for_thresholds2,predictions_for_thresholds3,predictions_for_thresholds4,g
        round_truth_for_thresholds1,ground_truth_for_thresholds2,ground_truth_for_thresholds3,ground_truth_for_thresholds4
```

References