UCLA

ECE 219 - Large-Scale Data Mining: Models and Algorithms
Winter 2021

Project 3: Collaborative Filtering

Authors:
Swapnil Sayan Saha (UID: 605353215)
Grant Young (UID: 505627579)

Question 1:

In this question, we are asked to report the sparsity of the movie rating information in the MovieLens dataset. The dataset contains 100836 ratings (in the range 0.5 to 5) for 9742 movies across 610 users. The sparsity is defined as:

$$S = \frac{R}{U \times M}$$

where, R = Total number of available ratings, U = Total number of users, M = Total number of movies. $U \times M$ gives the total number of possible ratings.

The resulting sparsity is 0.016999683055613623. This indicates that the ratings matrix is sparse, where the rows denote users and the columns denote movies, and each element denotes the rating of a movie provided by a user. This is expected because not every user can rate all the movies in the dataset.

Question 2:

In this question, we are asked to investigate the shape of the histogram of rating values. Figure 1 shows the histogram illustrating the frequency of the rating values.

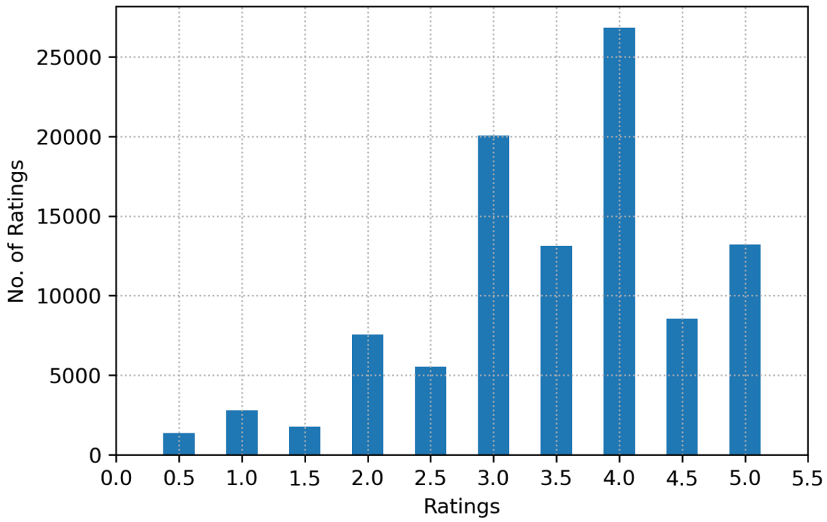


Figure 1: Histogram showing number of ratings for each rating interval.

From the histogram, we see that most users have rated movies higher up the ratings list, particularly between 3.0 to 5.0. In other words, the distribution of ratings is skewed to the left. We can explain this intuition by the fact that most viewers watch movies after looking at reviews from well-known ratings websites and sources as well hype generated around the movie. As a result, viewers usually only watch movies that they think they will like, and consequently, the user-ratings are higher as well. In addition, integer ratings (1, 2, 3, 4 and 5) have higher counts than fractional ratings (e.g. 0.5, 1.5, 3.5 etc.), which is also expected as most humans think of numbers as integers rather than fractional values.

Question 3:

In this question, we are asked to investigate the relationship between the number of ratings received and the movie index. Figure 2 shows the plot of the number of ratings versus movie index, with the movie receiving the largest number of ratings indexed as 1.

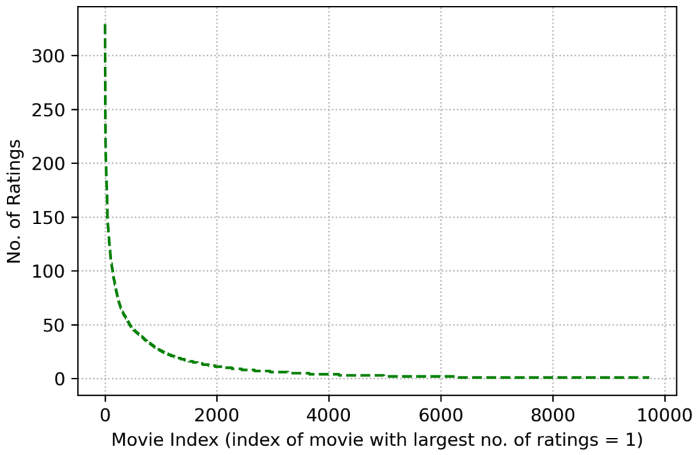


Figure 2: Plot showing number of ratings of each movie vs. index of movie (decreasing frequency).

From Figure 2, we see that the curve is monotonically decreasing, with approximately 500 movies (out of 9742) receiving more than 50 unique user ratings. This explains the sparsity of the ratings matrix, with only a few movies receiving multiple unique ratings.

Question 4:

In this question, we are asked to investigate the number of ratings given by each user. Figure 3 shows the plot of the number of ratings given by each user, with the user providing the largest number of ratings indexed as 1.

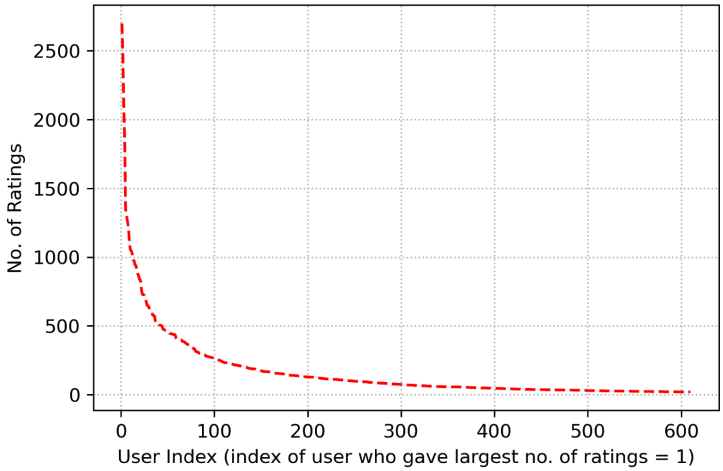


Figure 3: Plot showing number of ratings provided vs. index of user (decreasing frequency).

From Figure 3, we see that the curve is monotonically decreasing, with less than 50 users (out of 610) providing ratings to 500 movies or more (out of 9742). Coupled with the plot in Figure 2, this also explains why the ratings matrix is sparse, as a vast number of users do not provide sufficient number of unique ratings.

Question 5:

From Figure 2, we observe that curve is monotonically decreasing, with approximately 500 movies (out of 9742) receiving more than 50 unique user ratings. This indicates a vast majority of the movies did not receive sufficient unique user ratings, which explains the sparsity of the ratings matrix, with only a few movies receiving multiple unique ratings. This is because most viewers watch movies after looking at reviews from well-known ratings websites and sources as well hype generated around the movie by other users, which means that only a few movies are watched by a large audience. From a machine learning perspective, data sparsity is not coveted. As data moves into higher dimensions, the rapid increase in volume causes the data to become sparse in each dimension, causing the amount of data required for accurate classification and representation to increase exponentially for each dimension or feature. This is referred to as the "Curse of Dimensionality". However, since most of the elements in the sparse representation are 0, these elements contribute little to no information for the model being trained on the representation, resulting in a model with a large number of parameters that perform poorly on those movies with a low number of ratings due to lack of sufficient ratings and overfitting on those movies with a higher number of user ratings. A common technique to address this problem is to use regularization to encourage generalization and prevent formation of ill-conditioned classifiers, leading to a simpler model with lower number of weights.

Question 6:

In this question, we are asked to investigate the shape of the histogram of variance in rating values for each movie. Figure 4 shows the histogram illustrating the variance in rating values.

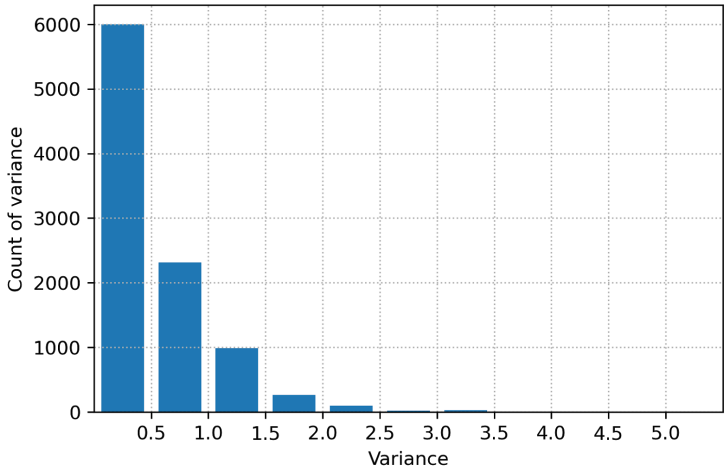


Figure 4: Distribution of variance in rating for each movie

From Figure 4, we see that the distribution of variance is skewed to the right, with a large number of movies having a variance between 0 and 1.5 (low variance). This means that the ratings provided to a certain movie do not vary wildly depending on the user and are consistent across all the users. This is expected because most viewers watch movies after looking at reviews from well-known ratings websites and sources as well hype generated around the movie by other users, resulting in similar ratings from all users for popular movies (from Figure 1). This is also evident in the plot in Figure 1, which shows that most ratings huddle in the 3 to 5 range.

Question 7:

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

Question 8:

$I_u \cap I_v$ indicates the set of movies (item indices) for which ratings have been specified for both user *u* and user *v*. $I_u \cap I_v$ can be null for the MovieLens dataset because there may be some movies that have been rated by user *u* but not user *v* and vice versa.

Question 9:

Centering the raw ratings around the mean ratings of the users reduces (normalizes) the user-specific bias and traits in the ratings, as well as variance caused by polar or critical (outlier) ratings by certain users. For example, some users might provide ratings at the higher or lower end of the rating spectrum (e.g., between 4 and 5 or between 1 and 3) while other users might mix and match ratings within the entire range of the spectrum (e.g., between 1 and 5). Mean centering helps remove these traits and outliers to make the data less noisy. Since we want to find the interaction between user ratings in the prediction function, mean-centering helps remove multicollinearity between the predictor variables, making it easier to find the significance or meaning of individual user ratings.

Question 10:

In this question, we are asked to design a k-nearest neighbors (k-NN) collaborative filter (CF) to predict the ratings of the movies in the MovieLens dataset and explore the relationship between the prediction error and number of neighbors. CF uses past ratings across users for similar items to predict items that a user might be interested in by exploiting the correlation of ratings across users or items from a sparse matrix of user ratings. k-NN is a neighborhood-based CF model, which uses behavior of similar users or behavior of users on similar items to recommend new items to users. There are two types of neighborhood-based CF model:

- *User-based*: CF that exploits similarity across users to predict ratings on new items.
- *Item-based*: CF based on the hypothesis that similar items are rated similarly by the same user.

For this project, we are asked to use user-based CF with k-NN model. The steps to use the k-NN user-based CF are as follows:

- To predict the rating of an item by a user, find the top *k* similar users to the target user who have also rated the item.
- Use the weighted average of the ratings of the *k* similar users to predict ratings for the target user on target item. Mathematically, the prediction function is given by:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_i} \text{sim}_{uv}(r_{vj} - \mu_v)}{\sum_{v \in P_i} \text{sim}_{uv}}, \qquad \mu_x = \frac{\sum_{k \in I_x} r_{xk}}{|I_x|}$$

  Here, P_i indicates the set of top *k* similar users who have rated the target item, *u* is the target user, *v* are other users in P_i, r_{vj} provides the rating of user *v* on item *j*. I_x denotes the set of items for which ratings have been provided by user *x*, while μ_x provides the mean rating for user x computed using all the ratings (useful to mean centering).
- To compute the similarity, we are asked to use the Pearson similarity metric (also known as centered cosine), which is given as:

$$\text{Pearson}(u, v) = \frac{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u)(r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u)^2} \sqrt{\sum_{k \in I_u \cap I_v} (r_{vk} - \mu_v)^2}}$$

  We do not use Euclidean distance to compute similarity because Euclidean distances to converge to a constant value between all sample points in higher dimensions for sparse ratings matrix, which can degrade recommendation performance. In addition, the centered cosine similarity metric takes into account the varying baselines (user-specific bias, traits and polar ratings) and attempts to normalize these differences.

We use the Surprise-Scikit library's `KNNWithMeans()` for this question to implement the K-NN user-based CF on the movie rating information, while using the same library's `cross_validate()` function to perform 10-fold cross-validation. Figure 5 shows the average prediction root mean squared error (RMSE) and mean absolute error (MAE) for various number of neighbors (top *k* users) with Pearson similarity metric (`sim_options={'name':'pearson'}` in the code).

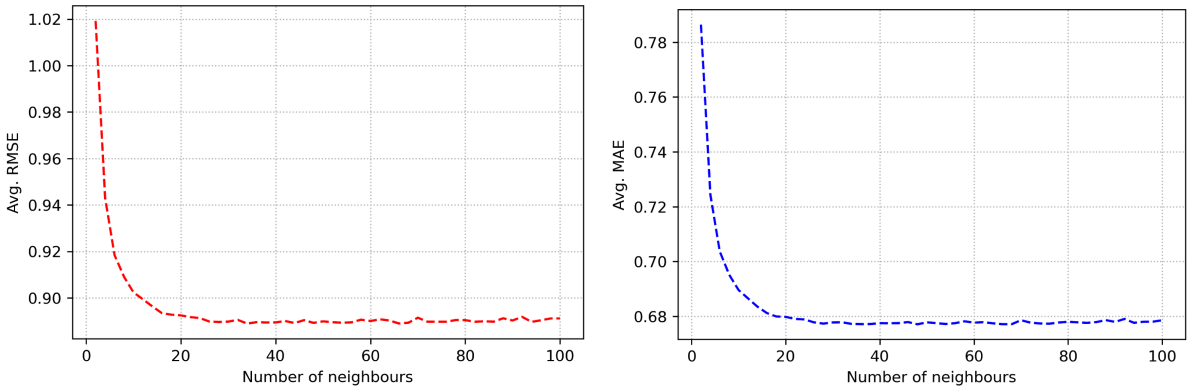


Figure 5: (Left) Average RMSE vs. *k* for K-NN user-based CF (Right) Average MAE vs. *k* for K-NN user-based CF

From Figure 5, we see that as the number of users in the neighborhood increases, the prediction error drops monotonically until a critical point, after which increasing *k* does not result in a significant decrease in prediction error. With more neighbors, the error is expected to decrease (or level out later) as the k-NN CF has more users within the vicinity to compare the ratings with.

Question 11:

In this question, we are asked to find the minimum value of *k*, beyond which the errors for k-NN user-based CF level out. Judging from the curves in Figure 5, we see that this occurs at ***k* = 20**, with the **steady-state RMSE = 0.89** and **steady-state MAE = 0.68**.

Question 12:

In this question, we are asked to design a k-NN user-based CF, with predictions being made on the popular movie trimmed test set, which contains only those movies that have received at least 2 ratings. We use the `KFold()` function in Surprise-Scikit library to split the dataset into 10 pairs of train and test folds and remove those movies from the test set that have less than 2 ratings. We test the CF with Pearson similarity metric for various number of neighbors, training on the training folds and testing on the trimmed set. We calculate the RMSE for each test fold and average. Figure 6 shows the average prediction RMSE for various number of neighbors (users).

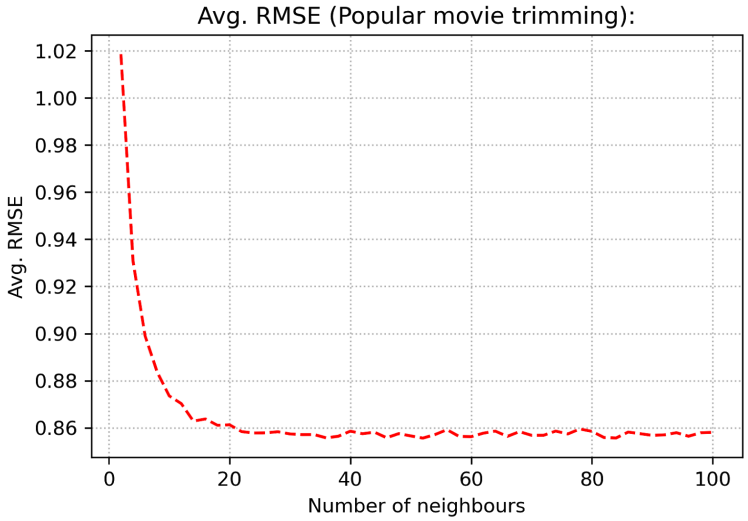


Figure 6: Average RMSE vs *k* for K-NN user-based CF on popular movie trimmed test set.

The minimum average RMSE for k-NN user-based CF on popular movie trimmed test set is 0.85556. From Figure 6, we see that the trend of monotonically decreasing prediction error continues similar to Figure 5, but with lower prediction error compared to testing on the entire dataset. This is expected because trimming out movies with fewer ratings removes "outliers", which can be difficult to evaluate by the prediction function due to lack of enough ratings or neighbors (users) for those movies.

Question 13:

In this question, we are asked to design a k-NN user-based CF, with predictions being made on the unpopular movie trimmed test set, which contains only those movies that have received less than or equal to 2 ratings. Figure 7 shows the average prediction RMSE for various number of neighbors (users).

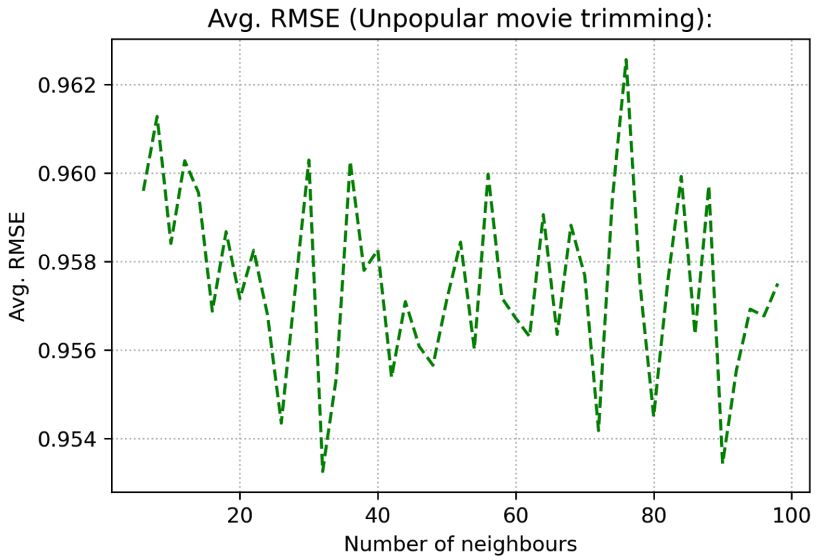


Figure 7: Average RMSE vs *k* for K-NN user-based CF on unpopular movie trimmed test set.

The minimum average RMSE for k-NN user-based CF on unpopular movie trimmed test set is 0.953259. This is much higher than the minimum average RMSE for popular movie trimmed test set and even the untrimmed set (Question 11), while the RMSE vs *k* curve becomes non-monotonic (increasing *k* does not result in lower prediction error) in nature with erratic jumps across neighbors. This is expected because the test set is now full of outliers containing only those movies with insufficient number of ratings. Since the predictor was trained on the entire dataset which also contains popular movies, the predictor has difficulty in correctly estimating the ratings of the "rare" items as it does not find enough users within the neighborhood who have rated the movies. As a result, the error is now effectively decoupled from *k* in a negative way and varying it does not result in any predictable change in average RMSE.

Question 14:

In this question, we are asked to design a k-NN user-based CF, with predictions being made on the high-variance movie trimmed test set, which contains only those movies that either has a variance of at least 2 or has at least 5 ratings. Figure 8 shows the average prediction RMSE for various number of neighbors (users).

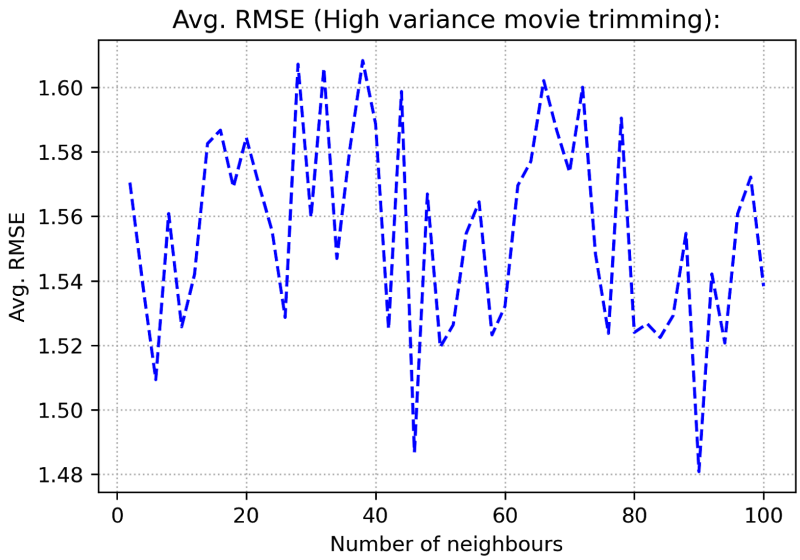


Figure 8: Average RMSE vs *k* for K-NN user-based CF on high-variance movie trimmed test set.

The minimum average RMSE for k-NN user-based CF on high-variance movie trimmed test set is 1.48077. This is worse than the prediction error in both the popular and unpopular movie trimmed test set, and the curve shows non-monotonic behavior similar to what was observed in Figure 7. With the test set containing only movies with erratic ratings, the predictor (which was trained to generalize on the entire training set) provides softer ratings for the movies with polar ratings, which results in a large prediction error that is effectively decoupled from the number of neighbors. Since the ratings are polar and vary wildly, the average ratings of each movie across neighbors are sensitive to outliers, resulting in no improvement in prediction error with increasing number of neighbors.

Question 15:

In this question, we are asked to plot the ROC curves for k-NN user-based CF for best value of *k* found in Question 11, which was 20, for thresholds [2.5, 3, 3.5, 4]. The ROC curve shows the plot for true-positive (TP) rate vs false-positive (FP) rate. For recommender systems, it is a measure of the relevance of the items recommended to the user.

To convert the predicted labels to binary, we set the predicted labels to 1 if it exceeded the threshold and 0 otherwise. We used `train_test_split()` from Surprise-Scikit to split the dataset into 90% training and 10% testing, while using `roc_curve()` and `auc()` from SciKit-Learn to get the TPR, FPR and area-under-curve (AUC) for each threshold. Figure 9 shows the ROC curves for the four thresholds for k-NN user-based CF with 20 neighbors (users).

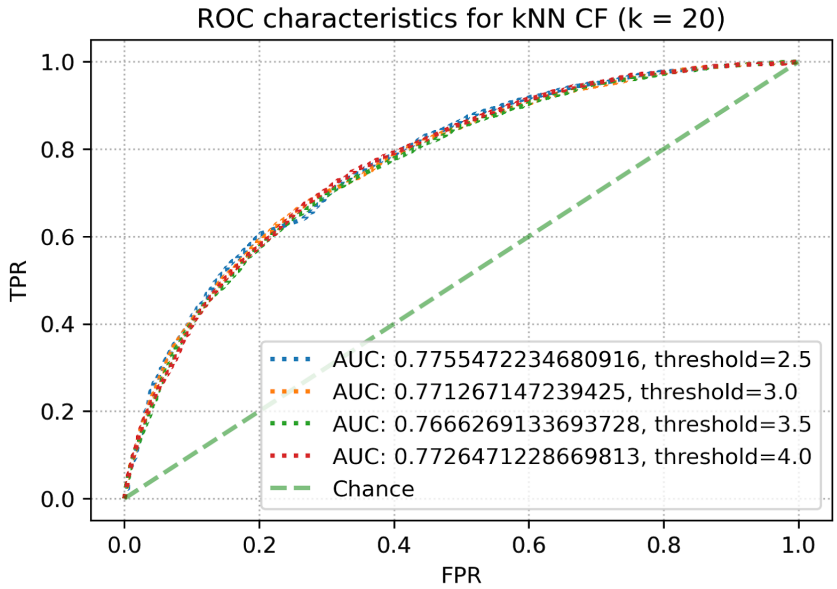


Figure 9: ROC curves for various thresholds for k-NN user-based CF with 20 neighbors.

The AUC for each threshold are as follows:

- **Threshold: 2.5, AUC: 0.775547**
- **Threshold: 3.0, AUC: 0.771267**
- **Threshold: 3.5, AUC: 0.766269**
- **Threshold: 4.0, AUC: 0.772647**

We see that as threshold increases, the AUC starts to decrease and then increases again for threshold = 4.0.

Question 16:

The optimization problem is not jointly convex for user latent space, U and item embedding space, V because of the existence of multiple local minima in the objective function gradient plane. This is because the matrix factorization model predicts ratings using the product of U and V, which does not satisfy the property of convexity as the objective function is permutation and rotation invariant.

The optimization problem can be solved using alternating least-squares (ALS), keeping U fixed and solving for V and vice versa in the next step. For fixed U, the least-squares formulation (without regularization) of the objective function is given by:

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

$$V = (UU^T)^{-1}UR$$

where, R = ratings matrix.

Question 17:

In this question, we are asked to design a non-negative matrix factorization (NNMF/NMF) CF to predict the ratings of the movies in the MovieLens dataset and explore the relationship between the prediction error and the number of latent factors. NNMF-CF is a model-based CF, more specifically, a latent factor-based CF. Model-based CF, in general, are more robust against noise and sparsity in the ratings matrix, using deep-learning, clustering or matrix completion (latent-factor based CF) to predict user rating on new items without using the sparse ratings matrix R every time to make a prediction. This results in scalable, fast and generalizable recommendation system.

In matrix factorization-based CF, R is expressed in terms of user latent space, U and item embedding space, V (which are low dimensional hidden factors for items and users), which are not sparse. Matrix decomposition, which is a simple embedding model, can be reformulated as an optimization problem with the following objective function:

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

The 2nd and 3rd terms in the optimization problem act as regularizer to encourage generalization and prevent overfitting. W acts as a mask metadata matrix indicating where r_{ij} is missing. The optimization problem with W is called weighted matrix factorization. In NNMF, we set constraints on the sign of U and V:

$$U >= 0, V >= 0$$

As mentioned in Question 16, the optimization program is non-convex and can be solved using ALS, keeping U fixed and solving for V and vice versa in the next step. One can also use stochastic gradient descent (SGD) to solve the optimization problem, which is slightly slower and less stable (harder to handle unobserved entries) than ALS but does not rely on loss squares. The prediction function is given by:

$$\hat{r}_{ij} = \sum_{s=1}^{k} u_{is} \cdot v_{js}$$

We use the Surprise-Scikit library's `NMF()` for this question to implement the NNMF CF on the movie rating information. Figure 10 shows the average prediction root RMSE and MAE for various number of latent factors.

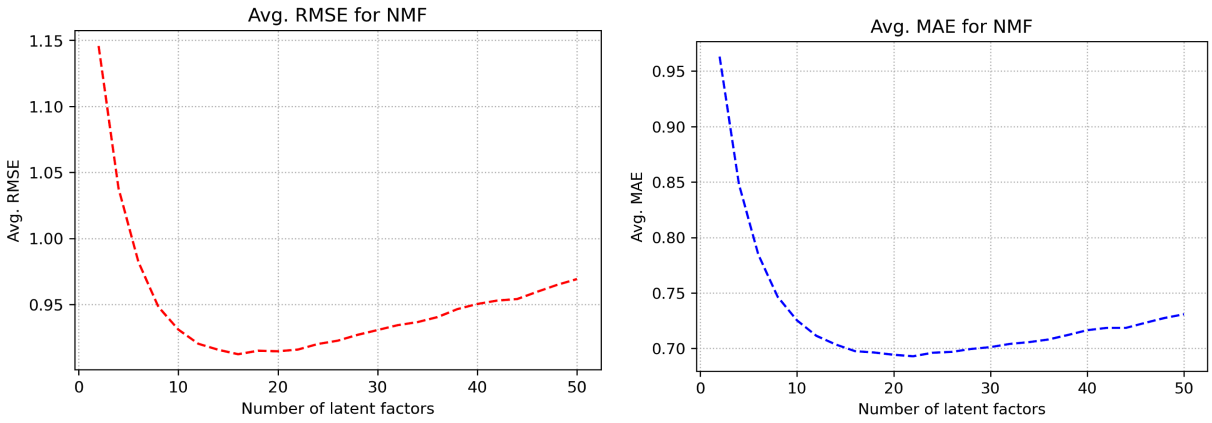


Figure 10: (Left) Average RMSE vs. *k* for NNMF CF (Right) Average MAE vs. *k* for NNMF CF

From Figure 10, we see that as the number of latent factors increases, the prediction error decreases sharply till a critical point, after which the prediction error starts to increase linearly. If the number of principal components (latent factors) is more, then the amount of semantic and complex information and structure in the data available for making predictions increases. This explains the initial improvement in prediction error. However, very large values of *k* indicate a high-dimensional and noisy ratings matrix, which degrades matrix completion performance. This is because in higher dimensions, the rapid increase in volume causes the factorized matrix to approach the original sparse matrix. Making predictions on the sparse matrix results in degradation of prediction error, which is explained by the rise in error in the later portions of the RMSE and MAE curves. In addition, NMF only allows positive entries in the reduced-rank embeddings, providing a shallow factorization with high information loss, and causing the decomposition depth of NMF to diminish in high-dimensions. Furthermore, NMF does not consider the geometry in the feature space basis and is non-unique and stochastic, with no guarantees of convergence to the optimal embeddings each time the function is called.

Question 18:

In this question, we are asked to find the optimum value of the number of latent factors for which the RMSE and MAE errors are minimal for NNMF CF. Judging from the curves in Figure 10, we see that this occurs at k = 16 for RMSE, with the **minimum average RMSE = 0.912293** and k = 22 for MAE, with the **minimum average MAE = 0.692898**. We decided to stick to $\boldsymbol{k = 16}$.

There are 19 genres in the MovieLens dataset, which is very close to the optimal value of k (which is 16). Note that the average of 16 and 22 is 19, which is exactly the number of movie genres.

Question 19:

In this question, we are asked to design an NNMF CF with predictions being made on the popular movie trimmed test set. Figure 11 shows the average prediction RMSE for various number of latent factors.

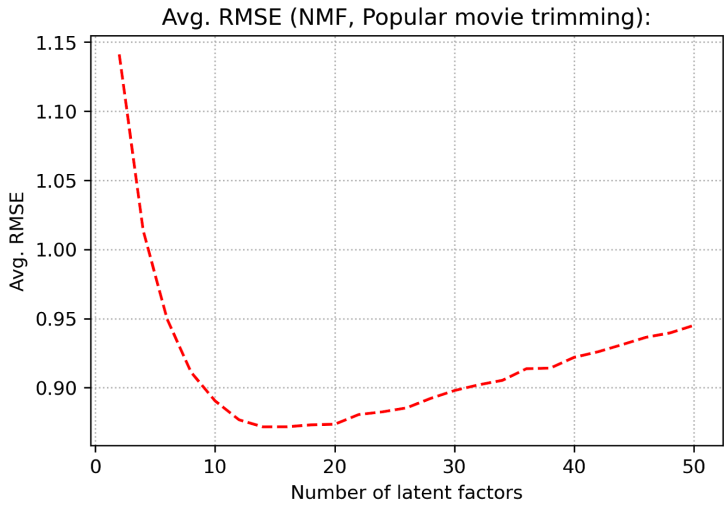


Figure 11: Average RMSE vs *k* for NNMF CF on popular movie trimmed test set.

The minimum average RMSE for NNMF CF on the popular movie trimmed test set is 0.871669. We have provided an explanation of the initially decreasing and then rising prediction error characteristics of the curve in Question 17.

Question 20:

In this question, we are asked to design an NNMF CF with predictions being made on the unpopular movie trimmed test set. Figure 12 shows the average prediction RMSE for various number of latent factors.

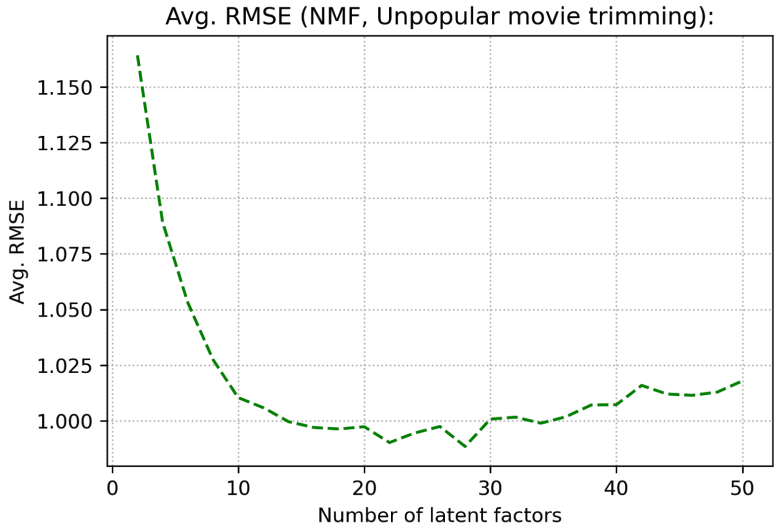


Figure 12: Average RMSE vs *k* for NNMF CF on unpopular movie trimmed test set.

The minimum average RMSE for NNMF CF on the unpopular movie trimmed test set is 0.988494. This is larger than the minimum average RMSE on the popular movie trimmed test set. This is expected because the test set is now full of outliers containing only those movies with insufficient number of ratings. Since the predictor was trained on the entire dataset which also contains popular movies, the predictor has difficulty in correctly estimating the ratings of the "rare" items as it did not have enough training data to correctly generate embeddings for those particular movies during training time. This also explains the erratic rise in the RMSE in the latter portion of the graph. In addition, the explanation for the initial drop and then rise of the curve has been provided in Question 17.

Question 21:

In this question, we are asked to design an NNMF CF with predictions being made on the high-variance movie trimmed test set. Figure 13 shows the average prediction RMSE for various number of latent factors.

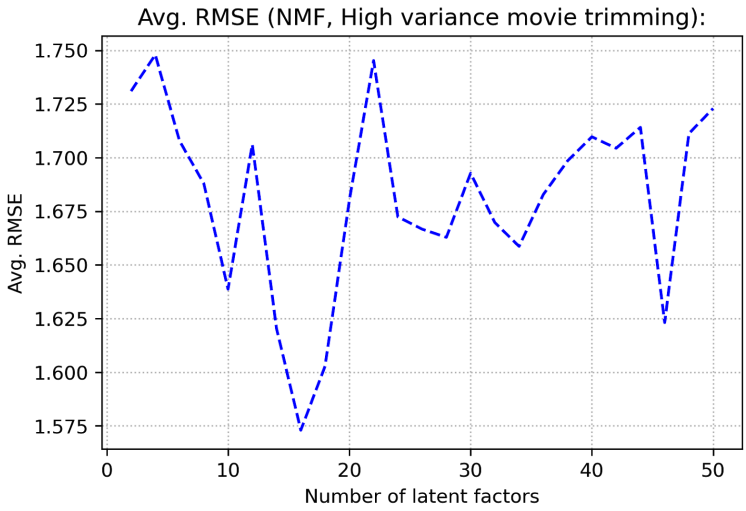


Figure 13: Average RMSE vs *k* for NNMF CF on high-variance movie trimmed test set.

The minimum average RMSE for NNMF CF on the high-variance movie trimmed test set is 1.572969. With the test set containing only movies with erratic ratings, the predictor (which was trained to generate generalizable embeddings on the entire training set) provides softer ratings for the movies with polar ratings, which results in a large prediction error that is effectively decoupled from the number of latent factors. Since the ratings are polar and vary wildly, the average ratings of each movie are sensitive to outliers and cannot be properly converted to low-dimensional embeddings or imputed. This results in worse average RMSE compared to popular and unpopular movie trimmed test set.

Question 22:

In this question, we are asked to plot the ROC curves for NNMF CF for best value of k found in Question 18, which was 16, for thresholds [2.5, 3, 3.5, 4]. Figure 14 shows the ROC curves for the four thresholds for NNMF CF with 16 latent factors.

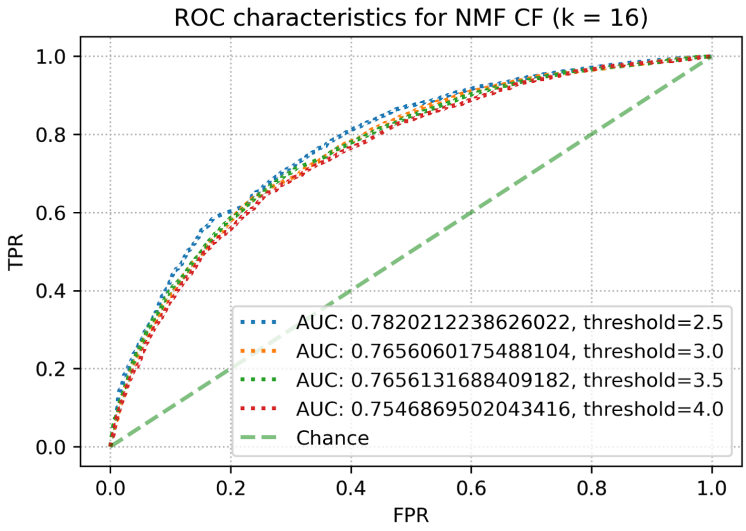


Figure 14: ROC curves for various thresholds for NNMF CF with 16 latent factors

The AUC for each threshold are as follows:

- **Threshold: 2.5, AUC: 0.782021**
- **Threshold: 3.0, AUC: 0.765606**
- **Threshold: 3.5, AUC: 0.765613**
- **Threshold: 4.0, AUC: 0.754686**

We see that as the threshold increases, the AUC decreases monotonically for NNMF CF.

Question 23:

In this question, we are asked to explore the connection between the latent factors and the movie genres to check the interpretability of NNMF CF. We obtained the V matrix using `nmf.qi` command, where `nmf = NMF(n_factors=20,n_epochs=50,verbose=False)`. We set k = 20 as asked by the question. Afterwards, we sorted the rows of V in descending order and printed out the top 10 movies for random columns of V. The results are as follows:

Column number of V: 1
Drama|Romance
Action|Adventure|Sci-Fi
Comedy|Drama|Romance
Drama
Comedy|Horror
Drama|Film-Noir|Mystery|Romance
Adventure|Drama|Western
Comedy|Drama
Comedy|Drama
Comedy|Drama

Column number of V: 3
Comedy|Drama
Sci-Fi
Crime|Drama|Thriller
Action|Crime
Comedy|Crime|Romance
Horror|Sci-Fi
Comedy|Romance
Action|Adventure|Fantasy
Comedy
Comedy|Romance

Column number of V: 5
Drama
Comedy
Children|Comedy
Horror|Mystery
Drama|Romance
Drama|Romance

Adventure|Children|Comedy
Action|Sci-Fi|Thriller|Western
Drama|Thriller
Comedy|Drama|Romance|Sci-Fi

Column number of V: 7
Drama
Drama|Mystery|Sci-Fi
Action|Crime|Drama|IMAX
Comedy
Children|Drama
Horror|Thriller
Action|Crime|Drama|Thriller
Comedy
Action|Comedy
Horror

Column number of V: 11
Drama
Comedy|Crime
Comedy
Drama
Comedy|Drama
Adventure|Drama
Comedy|Sci-Fi
Comedy
Action|Adventure|Sci-Fi
Adventure|Children|Comedy|Drama

Column number of V: 15
Comedy
Action|Comedy
Crime|Drama|Thriller
Comedy|Drama
Action|Drama|Thriller
Comedy
Children|Comedy
Drama|Thriller
Action|Adventure|Crime|Thriller

Crime|Drama|Thriller

Column number of V: 19

Comedy

Comedy

Documentary|Drama

Drama

Drama

Drama

Action|Adventure|Sci-Fi

Drama

Action|Horror|Sci-Fi

Adventure|Animation|Children|Comedy|Fantasy

From the genre list, we see that the top 10 movies belong to a small collection of genres. Each latent factor tends to group movies from small particular groups of genres, e.g. latent factor 19 seems to have most movies from the comedy and drama group, latent factor 5 seems to have movies from the drama|romance genres and latent factor 7 seems to have movies from the action, crime and horror genres.

Question 24:

In this question, we are asked to design a Matrix factorization with bias (MF-bias) CF to predict the ratings of the movies in the MovieLens dataset and explore the relationship between the prediction error and the number of latent factors. This is similar to NNMF CF optimization program but with bias terms on each user and item as follows:

$$\min_{U,V,b_u,b_i} \sum_{i=1}^{m} \sum_{j=1}^{n} W_{ij}(r_{ij} - (UV^T)_{ij})^2 + \lambda||U||_F^2 + \lambda||V||_F^2 + \lambda \sum_{u=1}^{m} b_u^2 + \lambda \sum_{i=1}^{n} b_i^2$$

The user bias term models user's traits to give polar ratings compared to the average, while the item bias term models how a certain item is rated compared to the average ratings. Integrating the bias information helps better model such user or movie-specific noise or outliers in the generated embeddings. The optimization program can be thought of as a singular value decomposition (SVD) problem. The prediction function is given by:

$$\hat{r}_{ij} = \sum_{s=1}^{k} (u_{is} \cdot v_{js}) + b_i + b_j + \mu$$

Here, μ is the mean of all ratings. We use the Surprise-Scikit library's `SVD()` for this question to implement the SVD CF on the movie rating information. Figure 15 shows the average prediction root RMSE and MAE for various number of latent factors.

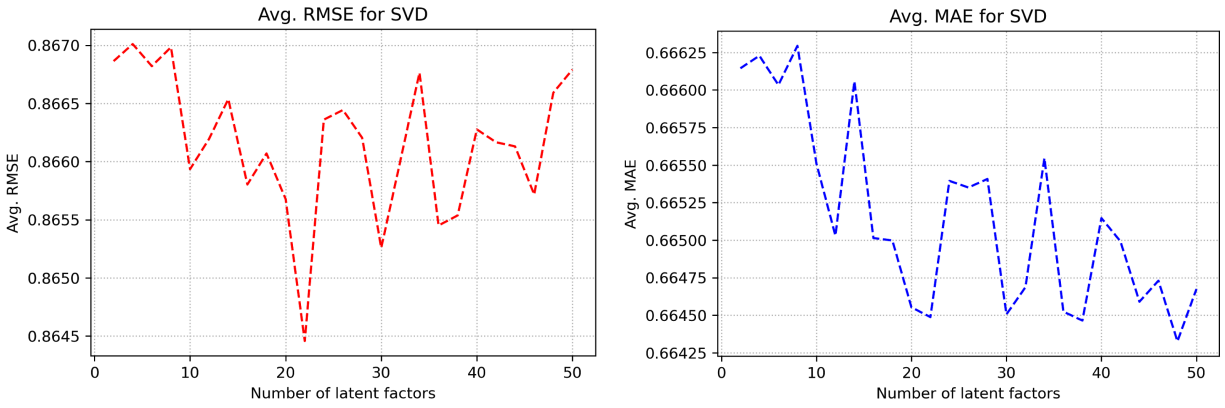


Figure 15: (Left) Average RMSE vs. *k* for SVD CF (Right) Average MAE vs. *k* for SVD CF.

From Figure 15, we see that the prediction error of SVD CF is very consistent within a very tiny range of error across all latent factors (do not get confused by the erratic shapes of the plots, the actual error is varying within a very tiny range if you notice the y-axis). In fact, SVD performs better than both k-NN and NNMF CF. This is expected because of several reasons:

- SVD is able to better represent the higher-dimensional feature matrix due to no constraints on U and V, providing a deep factorization with low information loss in both high and low dimensions.
- SVD produces a hierarchical and geometric basis ordered by relevance, producing embeddings with the most relevant features higher in the hierarchy. Thus, embeddings produced by SVD with higher values of k is not adding any significant distinguishable information, neither subtracting any important semantic information (due to noise) thanks to the ordering of the features.
- The embeddings produced by SVD are unique and deterministic.
- SVD takes into account user and movie-specific bias information and normalizes them appropriately to reduce sensitivity to outliers and noise.

Question 25:

In this question, we are asked to find the optimum value of the number of latent factors for which the RMSE and MAE errors are minimal for SVD CF (MF with bias). Judging from the curves in Figure 15, we see that this occurs at k = 22 for RMSE, with the **minimum average RMSE = 0.864458** and k = 48 for MAE, with the **minimum average MAE = 0.664326**. We decided to stick to $\boldsymbol{k = 22}$**,** as it is closer to the actual number of movie genres (19).

Question 26:

In this question, we are asked to design a SVD CF (MF with bias) with predictions being made on the popular movie trimmed test set. Figure 16 shows the average prediction RMSE for various number of latent factors.

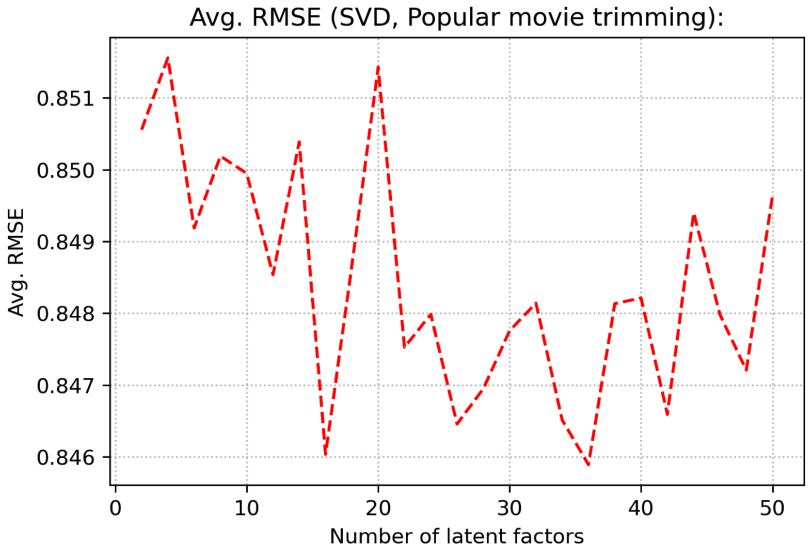


Figure 16: Average RMSE vs *k* for SVD CF on popular movie trimmed test set.

The minimum average RMSE for SVD CF on the popular movie trimmed test set is 0.845890. We have provided an explanation of the excellent consistency of SVD prediction error across all the latent factors in Question 24.

Question 27:

In this question, we are asked to design a SVD CF (MF with bias) with predictions being made on the unpopular movie trimmed test set. Figure 17 shows the average prediction RMSE for various number of latent factors.

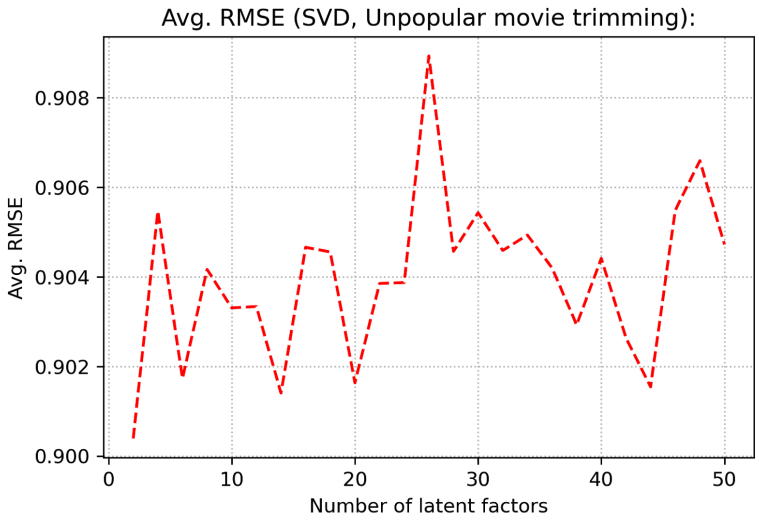


Figure 17: Average RMSE vs *k* for SVD CF on unpopular movie trimmed test set.

The minimum average RMSE for SVD CF on the unpopular movie trimmed test set is 0.90039. This is larger than the minimum average RMSE on the popular movie trimmed test set.

- The error is consistent across all the latent factors, which is explained in Question 24.
- In addition, the error is the lowest for unpopular movie trimmed set compared to NNMF or k-NN user-based CF. This is because of SVD's ability to model the bias for rarely rated items thanks to bias terms in the loss function, which is absent in k-NN or NNMF CF. As a result, SVD generates lower prediction errors when making inferences on outliers.
- The error is still however, larger than the popular movie trimmed set for SVD CF. The explanation which we provided for NNMF (Question 20) also holds here to explain why this happens.

Question 28:

In this question, we are asked to design a SVD CF (MF with bias) with predictions being made on the high-variance movie trimmed test set. Figure 18 shows the average prediction RMSE for various number of latent factors.

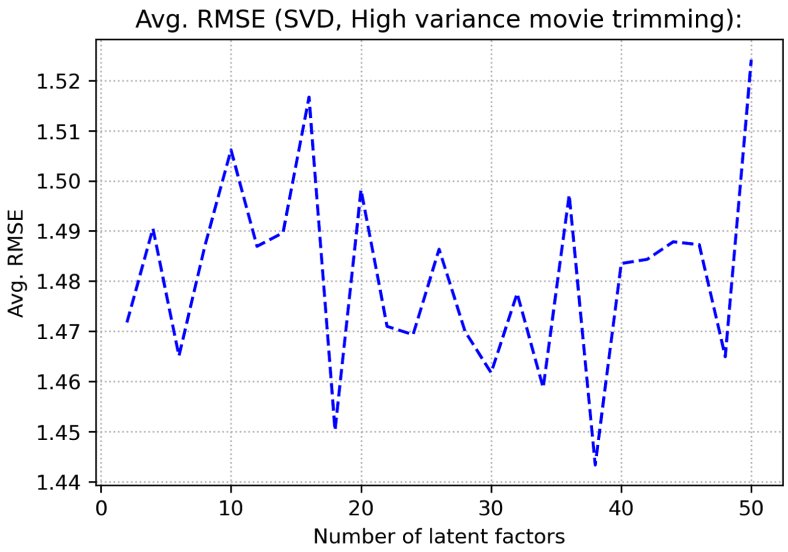


Figure 18: Average RMSE vs *k* for SVD CF on high-variance movie trimmed test set.

The minimum average RMSE for SVD CF on the high-variance movie trimmed test set is 1.44329. As expected, this is much larger than the prediction error made by SVD-CF on popular and unpopular movie trimmed set. We explained why this happens for NNMF in Question 21, and the same explanation holds.

Question 29:

In this question, we are asked to plot the ROC curves for SVD CF (MF with bias) for best value of k found in Question 25, which was 22, for thresholds [2.5, 3, 3.5, 4]. Figure 19 shows the ROC curves for the four thresholds for SVD CF with 22 latent factors.

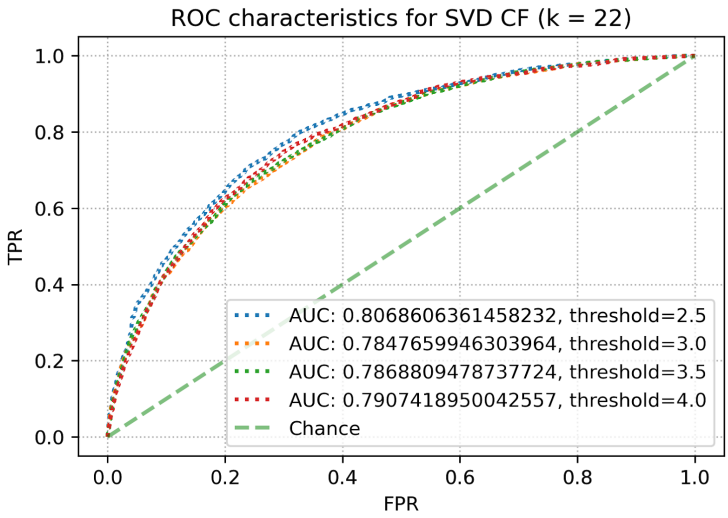


Figure 19: ROC curves for various thresholds for SVD CF with 22 latent factors

The AUC for each threshold are as follows:

- **Threshold: 2.5, AUC: 0.806860**
- **Threshold: 3.0, AUC: 0.784765**
- **Threshold: 3.5, AUC: 0.868809**
- **Threshold: 4.0, AUC: 0.790741**

We see that as the threshold increases, the AUC decreases monotonically for SVD CF until threshold = 4.0, at which point it starts increasing.

Question 30:

In this question, we are asked to design a naive CF to predict the ratings of the movies in the MovieLens dataset. The naive CF simply returns the mean ratings of a user on past items as the rating for the new item. In other words, the prediction function is given by:

$$\hat{r}_{ij} = \mu_i$$

There is no training involved in the naive CF process; it is just one-shot prediction on the test folds based on mean ratings on the entire dataset. **The average RMSE for 10 test folds was 0.934689**.

Question 31:

In this question, we are asked to design a naive CF with predictions being made on the popular movie trimmed test set. **The average RMSE for 10 test folds for naive CF was 0.924169 in this case**.

Question 32:

In this question, we are asked to design a naive CF with predictions being made on the unpopular movie trimmed test set. **The average RMSE for 10 test folds for naive CF was 0.954587 in this case**. This is slightly higher than the RMSE obtained on popular movie trimmed test set.

Question 33:

In this question, we are asked to design a naive CF with predictions being made on the high-variance movie trimmed test set. **The average RMSE for 10 test folds for naive CF was 1.466818 in this case**. This is much higher than the RMSE obtained on popular and unpopular movie trimmed test set.

Question 34:

In this question, we are asked to plot the ROC curves for SVD CF (MF with bias), k-NN user-based CF and NNMF CF for thresholds of 3. We used 20 neighbors for k-NN user-based CF, 22 latent factors for SVD (MF with bias) and 16 latent factors for NNMF, which are the best obtained in Question 11, 18 and 25. Figure 20 shows the ROC curves for all three CF.

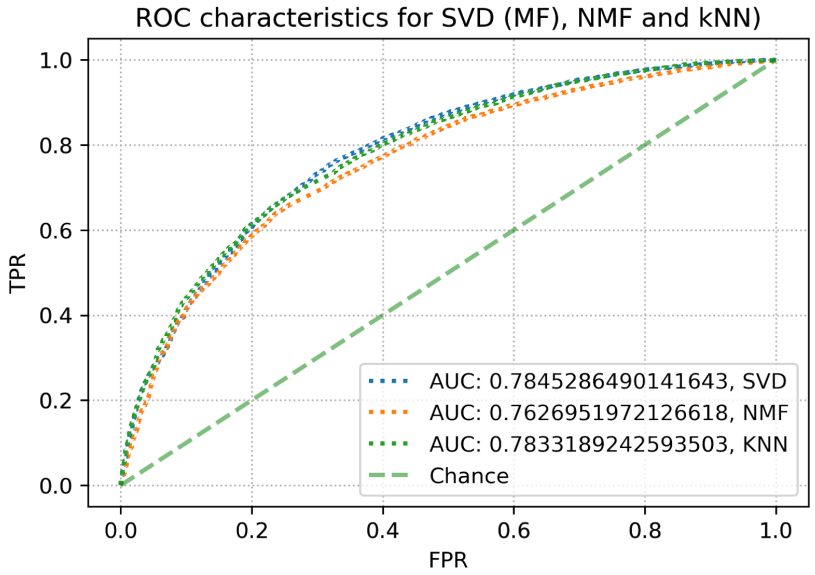


Figure 20: ROC curves for k-NN user-based CF (k = 20), NNMF CF (k = 16) and SVD CF (k = 22).

The AUC for each CF are as follows:

- **k-NN CF - 0.7833**
- **NNMF CF - 0.7626**
- **SVD CF - 0.7845**

From Figure 20, we can see that **SVD CF performs best among all the CF, followed by k-NN CF and NNMF-CF coming last**. We explain the performance as follows:

SVD vs NMF:

- SVD is able to better represent the higher-dimensional feature matrix due to no constraints on U and V, providing a deep factorization with low information loss. NMF on the other hand, restricts U and V to be positive and has fewer optimal choices of elements in U and V compared to SVD.
- SVD produces a hierarchical and geometric basis ordered by relevance, producing embeddings with the most relevant features and traits in the ratings matrix higher in the hierarchy. Thus, embeddings

produced by SVD are robust to outliers and noise in the ratings thanks to the ordering of the features. NMF, on the other hand, does not consider the geometry in the ratings matrix.

- The embeddings produced by SVD are unique and deterministic, whereas NMF is non-unique and stochastic, with no guarantees of convergence to the optimal U and V each time the function is called.
- SVD takes into account user and movie-specific bias information and normalizes them appropriately to reduce sensitivity to outliers and noise.

Why k-NN performs slightly worse than SVD:

- k-NN is not modeling the bias information separately for each user or item. As a result, it is more sensitive to outliers and rarely rated items.
- k-NN performs inference directly on the sparse ratings matrix, which yields poor prediction accuracy in high-dimensional space (curse of dimensionality). This also hurts the scalability of the recommender system. High-dimensional inference requires large amounts of training data to work properly, which is absent as the ratings matrix is sparse.
- k-NN is much less generalizable compared to latent-factor based models, as it cannot find semantic information and connections within the user-item ratings matrix while being sensitive to rarely rated items.

Question 35:

Precision is a measure of the exactness or reliability of a CF. Low precision indicates high number of false positives. In terms of recommendation systems, precision indicates the percentage of items the user actually liked out of the set of recommended items.

Recall, on the other hand, is a measure of sensitivity or completeness (information retrieval capacity) of a CF. Low recall indicates high number of false negatives. In terms of recommendation systems, recall indicates whether all the items the user likes were recommended to the user or not.

Question 36:

In this question, we are asked to show the plots of precision vs. *t*, recall vs. *t* and precision vs. accuracy curves for k-NN user-based CF with the best value of the number of neighbors found in Question 11, which is 20. *t* is the size of the set of items recommend to the user. In other words, we want to solve a ranking version of the recommender system. The steps undertaken to obtain the curves are as follows:

- Find the list of movies *G* liked by the user. The set only contains those movies that have a score greater than 3.
- Create a dictionary of all movies rated by each user.
- Split the dataset into 10 pairs of training set and test set.
- Drop those users in the test set who either rated less than *t* items (in the dictionary) or have no items in the set *G*.
- Compute predicted ratings with the chosen CF (in this case k-NN with Pearson similarity metric) for each user
- Sort the predicted ratings in descending order and select the first *t* items to yield $S(t)$.
- Calculate precision and recall for each user using the formula provided in the question.
- Take the mean of the precision and recall across as users and across all test folds.

Figure 21 shows all three curves for k-NN user-based CF for *t* ranging from 1 to 25.

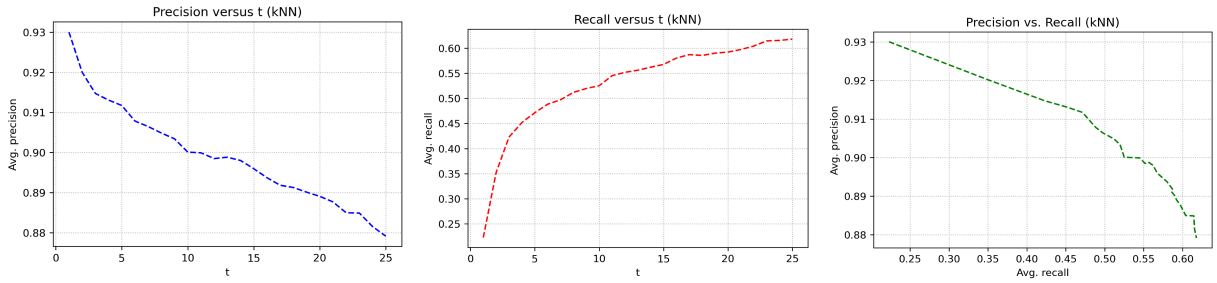


Figure 21: Precision vs *t*, Recall vs *t* and Precision vs Recall curves for k-NN user-based CF with 20 neighbors.

From Figure 21, we can make the following comments about the curves:

- As *t* increases, the average precision decreases. This is expected because the CF is more likely to generate false positives as the number of items recommended to the user increases. In other words, as $S(t)$ increases, the value of $S(t) \cap G$ is likely to drop as the CF is likely to suggest something that the user does not like. However, the reduction in precision is only around 5% for a $25\times$ increase in *t* for k-NN, so we can say that the precisions are overall consistent across *t*.
- As *t* increases, the average recall increases. This is because with a higher value of *t*, the probability that the predicted ratings will encompass all of the liked movies of the user (completeness) increases as well. The range of recall is much wider than that of the precision, with a 35% gap between the lowest recall

and highest recall for a $25\times$ increase in t for k-NN. This indicates that recall is more sensitive to how many items are being suggested compared to precision.

- As average recall increases, the average precision decreases. With a higher recall, apart from having a higher probability of including true positives in the recommendation list, the CF is also more likely to include items that are not liked by the user, which leads to a lower precision. In other words, a higher recall leads to a lower precision and vice versa.

Question 37:

In this question, we are asked to show the plots of precision vs. t, recall vs. t and precision vs. accuracy curves for NNMF CF with the best value of the number of latent factors found in Question 18, which is 16. The steps undertaken to obtain the curves are similar to the steps listed in Question 36. Figure 22 shows all three curves for NNMF CF for *t* ranging from 1 to 25.

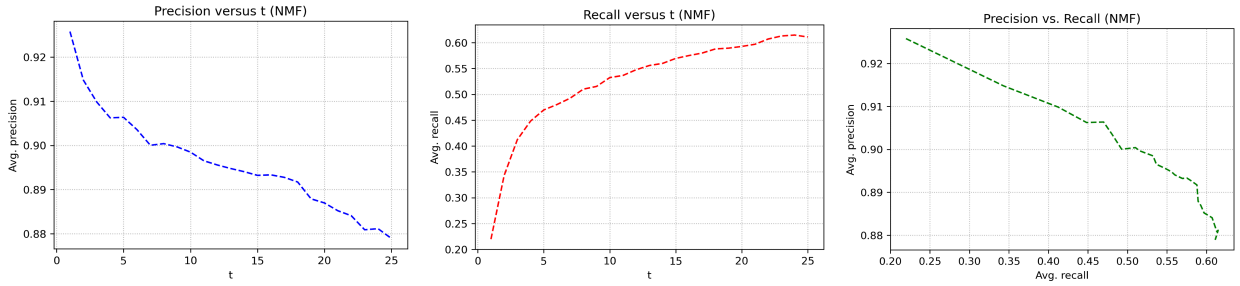


Figure 22: Precision vs *t*, Recall vs *t* and Precision vs Recall curves for NNMF CF with 16 latent factors

From Figure 22, we can make the following comments about the curves:

- As *t* increases, the average precision decreases. This is expected because the CF is more likely to generate false positives as the number of items recommended to the user increases. In other words, as $S(t)$ increases, the value of $S(t) \cap G$ is likely to drop as the CF is likely to suggest something that the user does not like. However, the reduction in precision is only around 4% for a $25\times$ increase in *t* for NNMF, so we can say that the precisions are overall consistent across *t*. This range is lower than what was obtained for k-NN.
- As *t* increases, the average recall increases. This is because with a higher value of *t*, the probability that the predicted ratings will encompass all of the liked movies of the user (completeness) increases as well. The range of recall is much wider than that of the precision, with a 35% gap between the lowest recall and highest recall for a $25\times$ increase in *t* for NNMF. This indicates that recall is more sensitive to how many items are being suggested compared to precision.
- As average recall increases, the average precision decreases. With a higher recall, apart from having a higher probability of including true positives in the recommendation list, the CF is also more likely to include items that are not liked by the user, which leads to a lower precision. In other words, a higher recall leads to a lower precision and vice versa.

Question 38:

In this question, we are asked to show the plots of precision vs. t, recall vs. t and precision vs. accuracy curves for SVD CF (MF with bias) with the best value of the number of latent factors found in Question 25, which is 22. The steps undertaken to obtain the curves are similar to the steps listed in Question 36. Figure 23 shows all three curves for SVD CF for *t* ranging from 1 to 25.

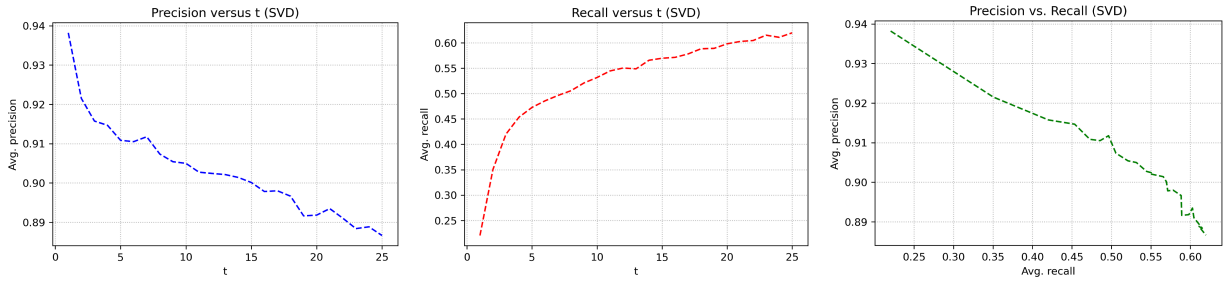


Figure 23: Precision vs *t*, Recall vs *t* and Precision vs Recall curves for SVD CF with 22 latent factors

From Figure 23, we can make the following comments about the curves:

- As *t* increases, the average precision decreases. This is expected because the CF is more likely to generate false positives as the number of items recommended to the user increases. In other words, as $S(t)$ increases, the value of $S(t) \cap G$ is likely to drop as the CF is likely to suggest something that the user does not like. However, the reduction in precision is only around 5% for a $25\times$ increase in *t* for SVD, so we can say that the precisions are overall consistent across *t*.
- As *t* increases, the average recall increases. This is because with a higher value of *t*, the probability that the predicted ratings will encompass all of the liked movies of the user (completeness) increases as well. The range of recall is much wider than that of the precision, with a 35% gap between the lowest recall and highest recall for a $25\times$ increase in *t* for SVD. This indicates that recall is more sensitive to how many items are being suggested compared to precision.
- As average recall increases, the average precision decreases. With a higher recall, apart from having a higher probability of including true positives in the recommendation list, the CF is also more likely to include items that are not liked by the user, which leads to a lower precision. In other words, a higher recall leads to a lower precision and vice versa.

Question 39:

In this question, we are asked to plot the precision vs. recall curves for k-NN user-based CF, NNMF CF and SVD CF (MF with bias) obtained in Questions 36, 37 and 38 on the same plot and compare their performance. Figure 24 shows the plot.

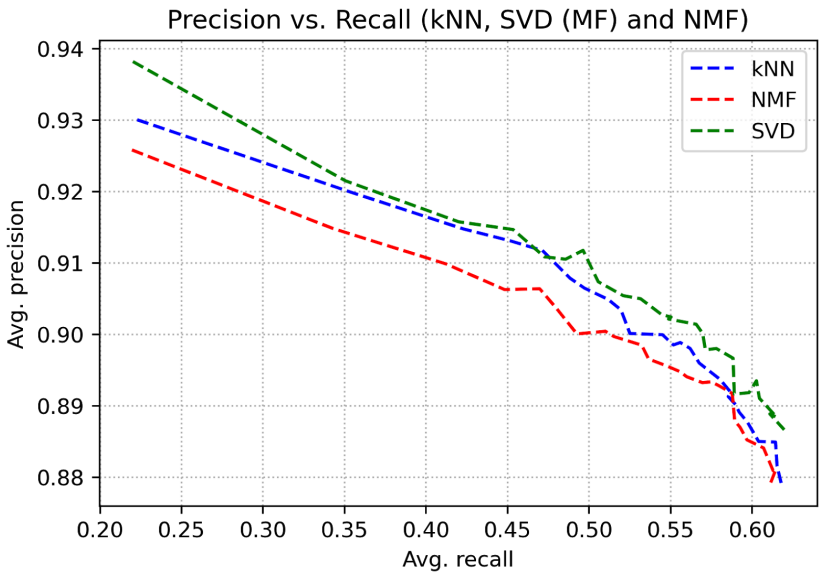


Figure 24: Precision vs Recall curves for k-NN, NNMF and SVD CF.

From Figure 24, we observe that SVD CF (MF with bias) provides the best performance as its precision drops slower with increase in recall compared to NNMF or k-NN CF while also maintaining a higher precision for each recall value compared to the latter two. k-NN user-based CF provides the 2nd best performance, followed by NMF-CF. In other words, we can say that SVD CF provides the most relevant recommended list (set of items most likely to be favored by the user) to each user, followed by k-NN CF and lastly NNMF-CF.

project_219_3_Saha_Young

February 16, 2021

[183]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
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
```

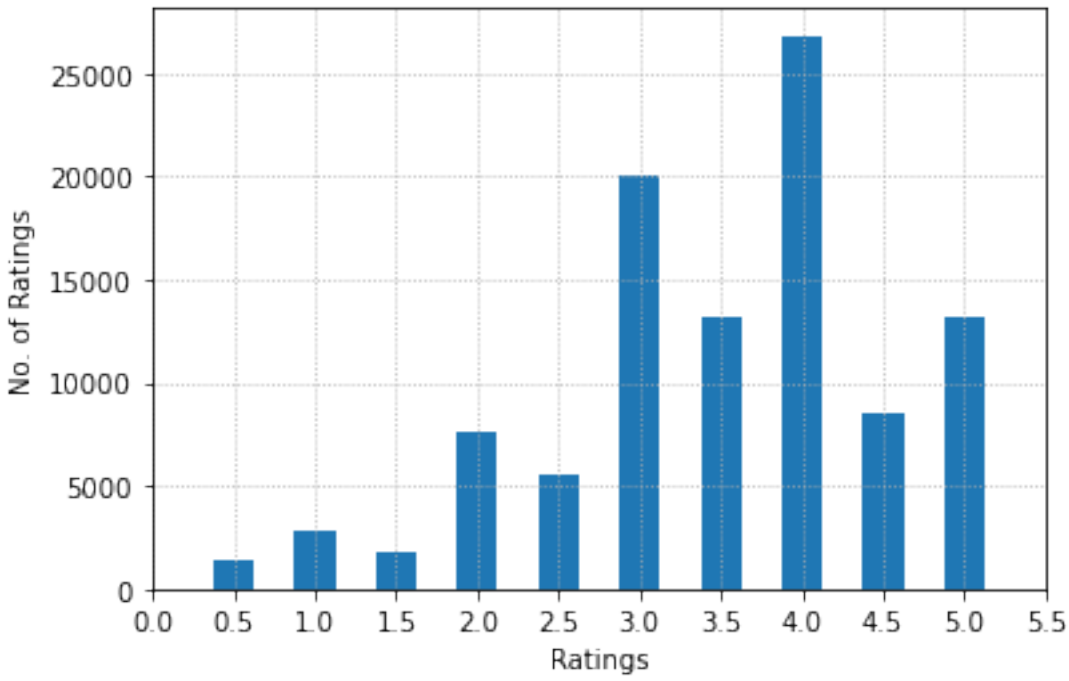
1 Question 1

[2]:
```
dataset_folder = 'movielinks/'
Ratings_file = pd.read_csv(dataset_folder+"ratings.
 ↪csv",usecols=['userId','movieId','rating'])
user_ID = Ratings_file.pop('userId').values
movie_ID = Ratings_file.pop('movieId').values
rating = Ratings_file.pop('rating').values
sparsity = len(rating)/(len(set(movie_ID))*len(set(user_ID)))
print('Sparsity:',sparsity)
```

```
Sparsity: 0.016999683055613623
```

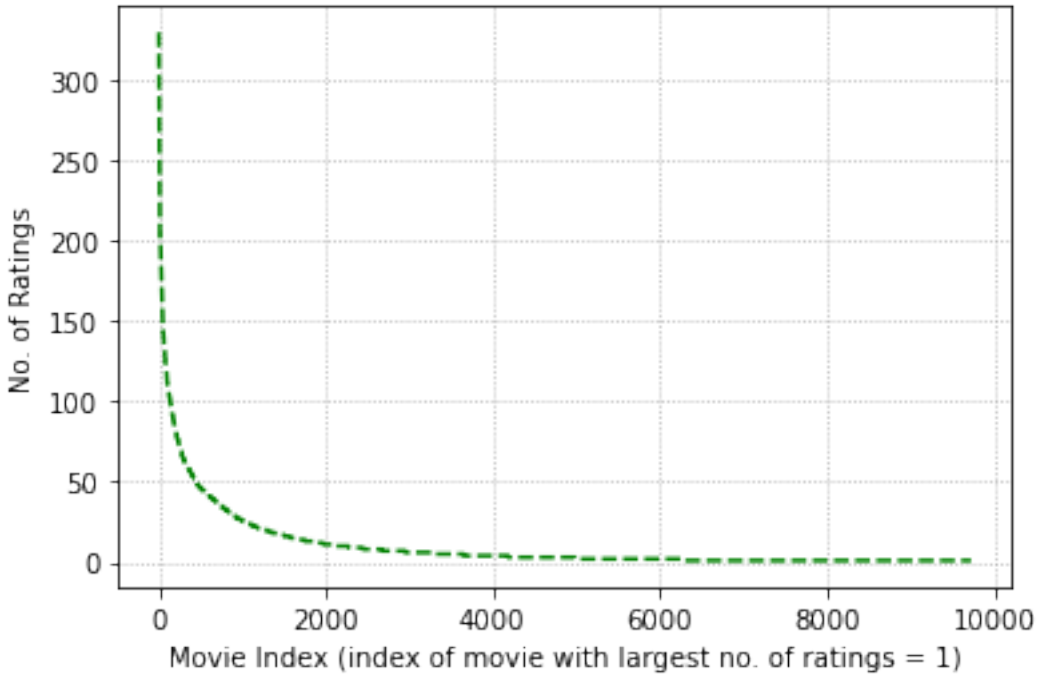
2 Question 2

[3]:
```
u, inv = np.unique(rating, return_inverse=True)
plt.bar(u, np.bincount(inv), width=0.25)
locs, labels = plt.xticks()
plt.grid(linestyle=':')
plt.xticks(np.arange(0,6,0.5),rotation=0)
plt.ylabel('No. of Ratings')
plt.xlabel('Ratings')
plt.savefig('Q2.png',dpi=300,bbox_inches='tight')
plt.show()
```



3 Question 3

[4]:
```
unique, counts = np.unique(movie_ID, return_counts=True)
plt.plot(range(1,len(unique)+1),counts[np.argsort(counts)[::
 ↪-1]],linestyle='--',color='g')
plt.grid(linestyle=':')
plt.ylabel('No. of Ratings')
plt.xlabel('Movie Index (index of movie with largest no. of ratings = 1)')
plt.savefig('Q3.png',dpi=300,bbox_inches='tight')
plt.show()
```

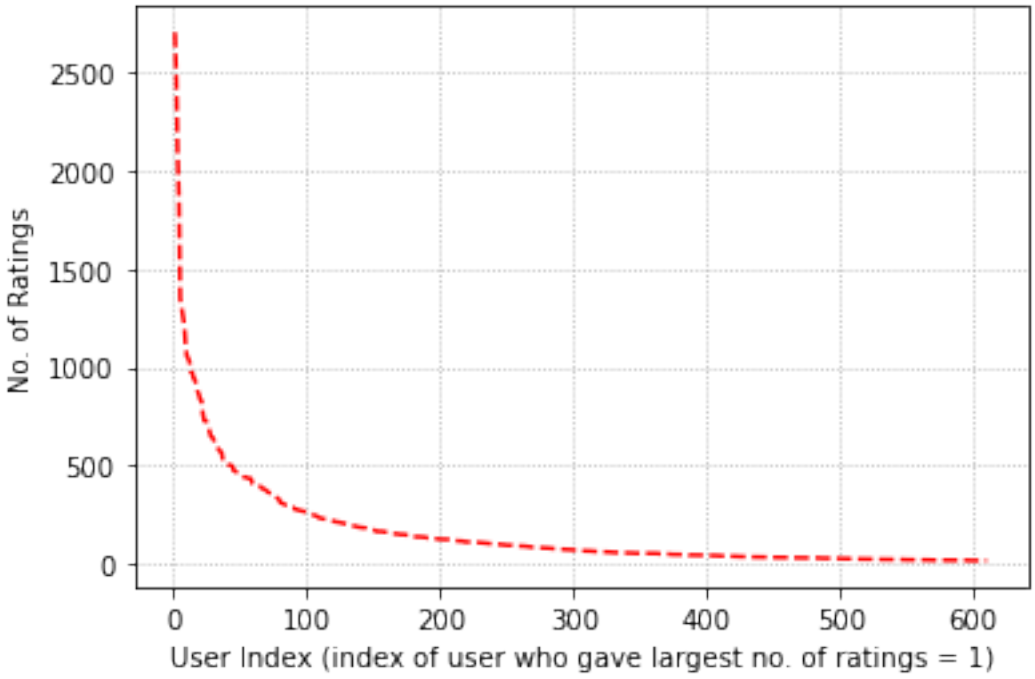


[5]:
```
movie_count_dict = {}
x = list(range(1,len(unique)+1))
for key in unique[np.argsort(counts)[::-1]]:
    for value in x:
        movie_count_dict[key] = value
        x.remove(value)
        break
print('Top 10 rated movies (Movie ID, Index):')
print(list(movie_count_dict.items())[0:10])
```

```
Top 10 rated movies (Movie ID, Index):
[(356, 1), (318, 2), (296, 3), (593, 4), (2571, 5), (260, 6), (480, 7), (110,
8), (589, 9), (527, 10)]
```

3.1 Question 4

[6]:
```
unique, counts = np.unique(user_ID, return_counts=True)
plt.plot(range(1,len(unique)+1),counts[np.argsort(counts)[::
 ↪-1]],linestyle='--',color='r')
plt.grid(linestyle=':')
plt.ylabel('No. of Ratings')
plt.xlabel('User Index (index of user who gave largest no. of ratings = 1)')
plt.savefig('Q4.png',dpi=300,bbox_inches='tight')
plt.show()
```



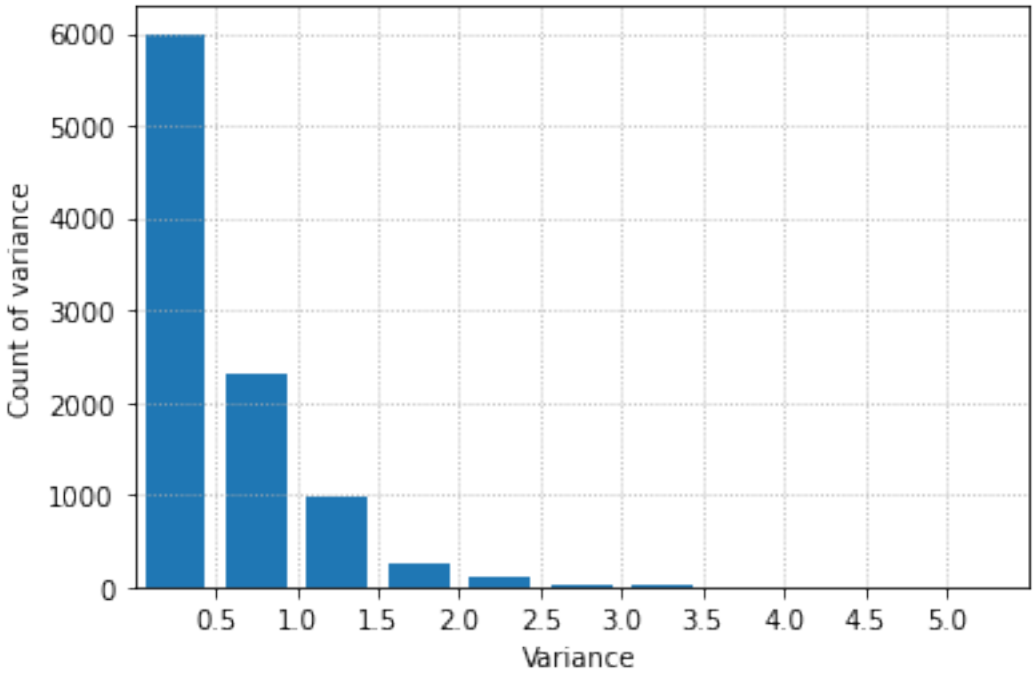
[7]:
```
user_count_dict = {}
x = list(range(1,len(unique)+1))
for key in unique[np.argsort(counts)[::-1]]:
    for value in x:
        user_count_dict[key] = value
        x.remove(value)
        break
print('Top 10 users who rated most number of times (User ID, Index):')
print(list(user_count_dict.items())[0:10])
```

```
Top 10 users who rated most number of times (User ID, Index):
[(414, 1), (599, 2), (474, 3), (448, 4), (274, 5), (610, 6), (68, 7), (380, 8),
(606, 9), (288, 10)]
```

4 Question 6

[8]:
```
unique_movie_ID = list(set(movie_ID))
movie_ID_list = []
var_list = []
for j in range(len(unique_movie_ID)):
    indices = [i for i, x in enumerate(movie_ID) if x == unique_movie_ID[j]]
    var = np.var(np.array(rating[indices]))
    movie_ID_list.append(unique_movie_ID[j])
    var_list.append(var)
```

[9]:
```
plt.hist(var_list, bins=np.arange(0,5.5,0.5),rwidth=0.75)
plt.xticks(np.arange(0.5,5.5,0.5))
plt.xlim([0, 5.5])
plt.grid(linestyle=':')
plt.xlabel('Variance')
plt.ylabel('Count of variance')
plt.savefig('Q6.png',dpi=300,bbox_inches='tight')
plt.show()
```



5 Question 10

[10]:
```
reader = Reader(line_format='user item rating␣
 ↪timestamp',sep=',',rating_scale=(0.5, 5),skip_lines=1)
ratings_dataset = Dataset.load_from_file(dataset_folder+"ratings.
 ↪csv",reader=reader)
```

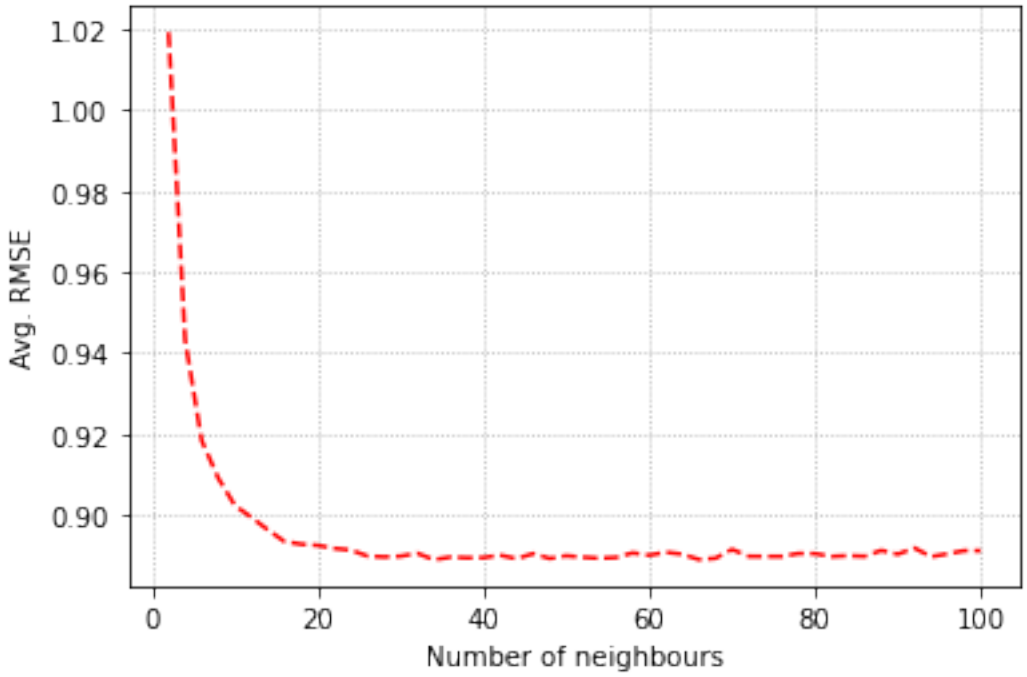
[11]:
```
k = np.arange(2,102,2)
rmse = []
mae = []
for item in k:
    print('Testing for k =',item)
    res = cross_validate(KNNWithMeans(k=item,sim_options={'name':'pearson'}),
```

```
                        measures=['rmse','mae'],data =␣
↪ratings_dataset,cv=10,n_jobs=-1)
   rmse.append(np.mean(res['test_rmse']))
   mae.append(np.mean(res['test_mae']))
```

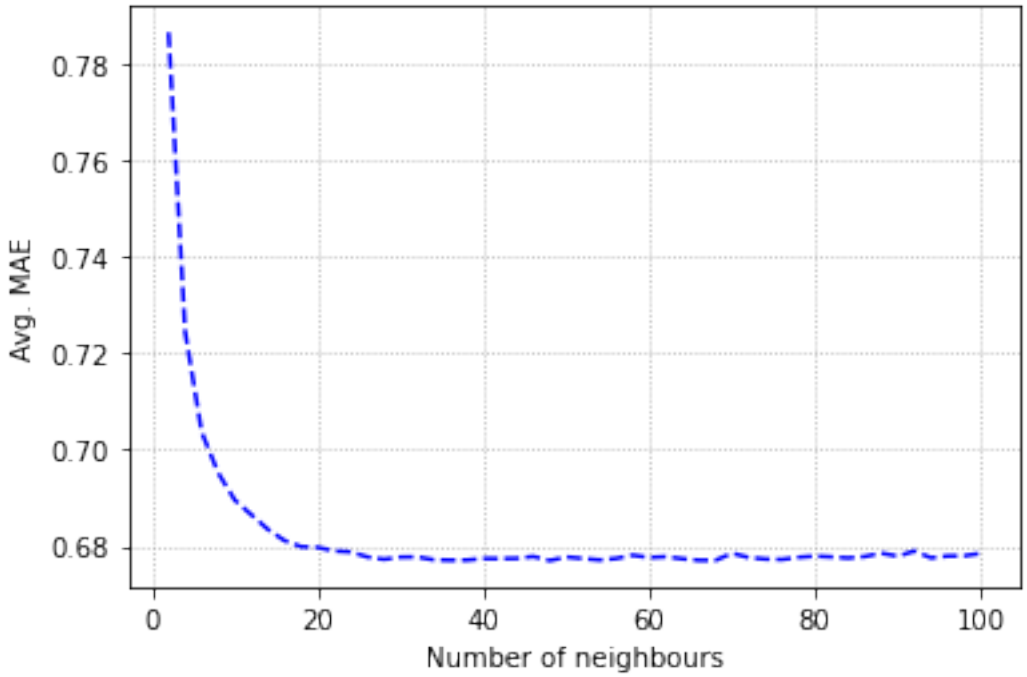
```
Testing for k = 2
Testing for k = 4
Testing for k = 6
Testing for k = 8
Testing for k = 10
Testing for k = 12
Testing for k = 14
Testing for k = 16
Testing for k = 18
Testing for k = 20
Testing for k = 22
Testing for k = 24
Testing for k = 26
Testing for k = 28
Testing for k = 30
Testing for k = 32
Testing for k = 34
Testing for k = 36
Testing for k = 38
Testing for k = 40
Testing for k = 42
Testing for k = 44
Testing for k = 46
Testing for k = 48
Testing for k = 50
Testing for k = 52
Testing for k = 54
Testing for k = 56
Testing for k = 58
Testing for k = 60
Testing for k = 62
Testing for k = 64
Testing for k = 66
Testing for k = 68
Testing for k = 70
Testing for k = 72
Testing for k = 74
Testing for k = 76
Testing for k = 78
Testing for k = 80
Testing for k = 82
Testing for k = 84
```

```
Testing for k = 86
Testing for k = 88
Testing for k = 90
Testing for k = 92
Testing for k = 94
Testing for k = 96
Testing for k = 98
Testing for k = 100
```

[12]:
```
plt.plot(k,rmse,linestyle='--',color='r')
plt.grid(linestyle=':')
plt.ylabel('Avg. RMSE')
plt.xlabel('Number of neighbours')
plt.savefig('Q10a.png',dpi=300,bbox_inches='tight')
plt.show()
```



[13]:
```
plt.plot(k,mae,linestyle='--',color='b')
plt.grid(linestyle=':')
plt.ylabel('Avg. MAE')
plt.xlabel('Number of neighbours')
plt.savefig('Q10b.png',dpi=300,bbox_inches='tight')
plt.show()
```



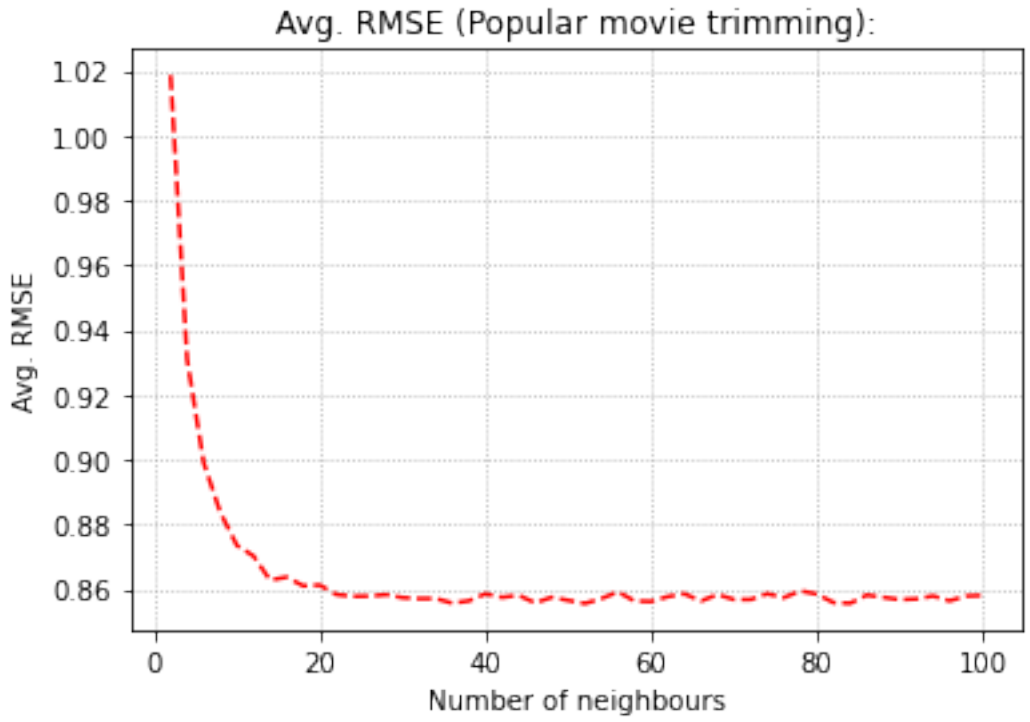
6 Question 12

[31]:
```
rmse_pop = []
kf =  KFold(n_splits=10)
for item in k:
    local_rmse = []
    print('Testing for k =',item)
    for trainset, testset in kf.split(ratings_dataset):
        trim_list = []
        unique, counts = np.unique([row[1] for row in testset],␣
↪return_counts=True)
        for i in range(len(counts)):
            if(counts[i]<=2):
                trim_list.append(unique[i])
        trimmed_set = [j for j in testset if j[1] not in trim_list]
        res = KNNWithMeans(k=item,sim_options={'name':'pearson'},verbose=False).
↪fit(trainset).test(trimmed_set)
        local_rmse.append(accuracy.rmse(res,verbose=False))
    rmse_pop.append(np.mean(local_rmse))
```

```
Testing for k = 2
Testing for k = 4
Testing for k = 6
Testing for k = 8
```

```
Testing for k = 10
Testing for k = 12
Testing for k = 14
Testing for k = 16
Testing for k = 18
Testing for k = 20
Testing for k = 22
Testing for k = 24
Testing for k = 26
Testing for k = 28
Testing for k = 30
Testing for k = 32
Testing for k = 34
Testing for k = 36
Testing for k = 38
Testing for k = 40
Testing for k = 42
Testing for k = 44
Testing for k = 46
Testing for k = 48
Testing for k = 50
Testing for k = 52
Testing for k = 54
Testing for k = 56
Testing for k = 58
Testing for k = 60
Testing for k = 62
Testing for k = 64
Testing for k = 66
Testing for k = 68
Testing for k = 70
Testing for k = 72
Testing for k = 74
Testing for k = 76
Testing for k = 78
Testing for k = 80
Testing for k = 82
Testing for k = 84
Testing for k = 86
Testing for k = 88
Testing for k = 90
Testing for k = 92
Testing for k = 94
Testing for k = 96
Testing for k = 98
Testing for k = 100
```

[32]:
```
plt.plot(k,rmse_pop,linestyle='--',color='r')
plt.grid(linestyle=':')
plt.title('Avg. RMSE (Popular movie trimming):')
plt.ylabel('Avg. RMSE')
plt.xlabel('Number of neighbours')
plt.savefig('Q12.png',dpi=300,bbox_inches='tight')
plt.show()
```



[33]:
```
print("Minimum avg. RMSE (Popular movie trimming):", min(rmse_pop))
```

```
Minimum avg. RMSE (Popular movie trimming): 0.855562336401092
```

6.1 Question 13

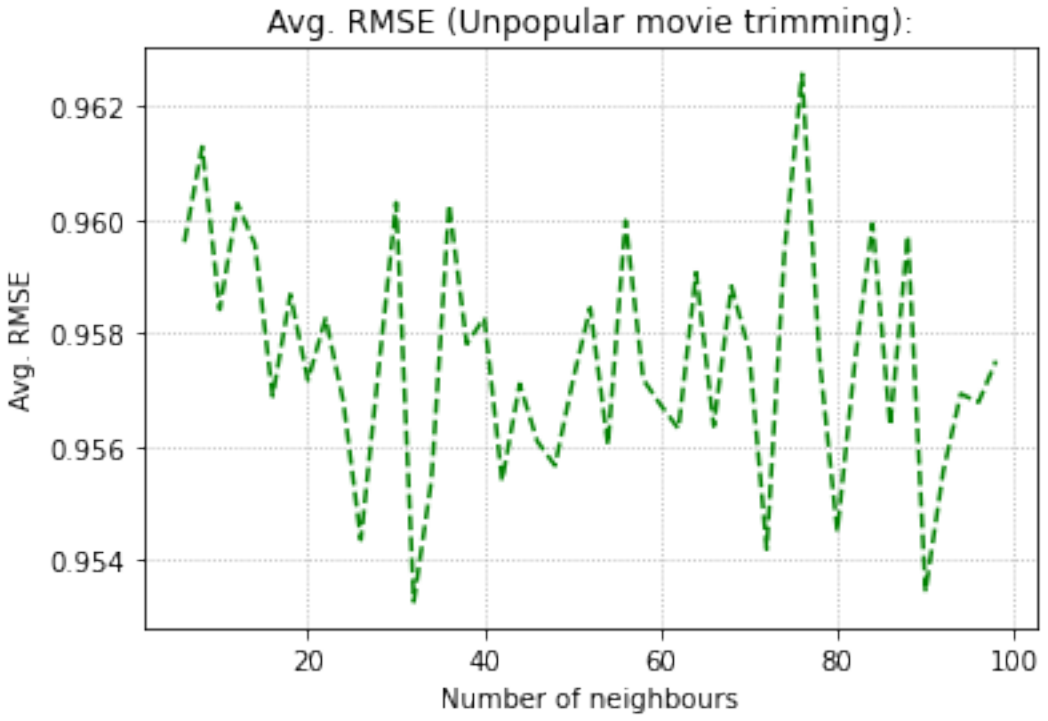
[34]:
```
rmse_unpop = []
kf =  KFold(n_splits=10)
for item in k:
    local_rmse = []
    print('Testing for k =',item)
    for trainset, testset in kf.split(ratings_dataset):
        trim_list = []
        unique, counts = np.unique([row[1] for row in testset],␣
 ↪return_counts=True)
```

```
        for i in range(len(counts)):
            if(counts[i]>2):
                trim_list.append(unique[i])
        trimmed_set = [j for j in testset if j[1] not in trim_list]
        res = KNNWithMeans(k=item,sim_options={'name':'pearson'},verbose=False).
↪fit(trainset).test(trimmed_set)
        local_rmse.append(accuracy.rmse(res,verbose=False))
    rmse_unpop.append(np.mean(local_rmse))
```

```
Testing for k = 2
Testing for k = 4
Testing for k = 6
Testing for k = 8
Testing for k = 10
Testing for k = 12
Testing for k = 14
Testing for k = 16
Testing for k = 18
Testing for k = 20
Testing for k = 22
Testing for k = 24
Testing for k = 26
Testing for k = 28
Testing for k = 30
Testing for k = 32
Testing for k = 34
Testing for k = 36
Testing for k = 38
Testing for k = 40
Testing for k = 42
Testing for k = 44
Testing for k = 46
Testing for k = 48
Testing for k = 50
Testing for k = 52
Testing for k = 54
Testing for k = 56
Testing for k = 58
Testing for k = 60
Testing for k = 62
Testing for k = 64
Testing for k = 66
Testing for k = 68
Testing for k = 70
Testing for k = 72
Testing for k = 74
Testing for k = 76
```

```
Testing for k = 78
Testing for k = 80
Testing for k = 82
Testing for k = 84
Testing for k = 86
Testing for k = 88
Testing for k = 90
Testing for k = 92
Testing for k = 94
Testing for k = 96
Testing for k = 98
Testing for k = 100
```

[62]:
```
plt.plot(k[2:-1],rmse_unpop[2:-1],linestyle='--',color='g')
plt.grid(linestyle=':')
plt.title('Avg. RMSE (Unpopular movie trimming):')
plt.ylabel('Avg. RMSE')
plt.xlabel('Number of neighbours')
plt.savefig('Q13.png',dpi=300,bbox_inches='tight')
plt.show()
```



[63]:
```
print("Minimum avg. RMSE (Unpopular movie trimming):", min(rmse_unpop))
```

```
Minimum avg. RMSE (Unpopular movie trimming): 0.9532593034140208
```

7 Question 14

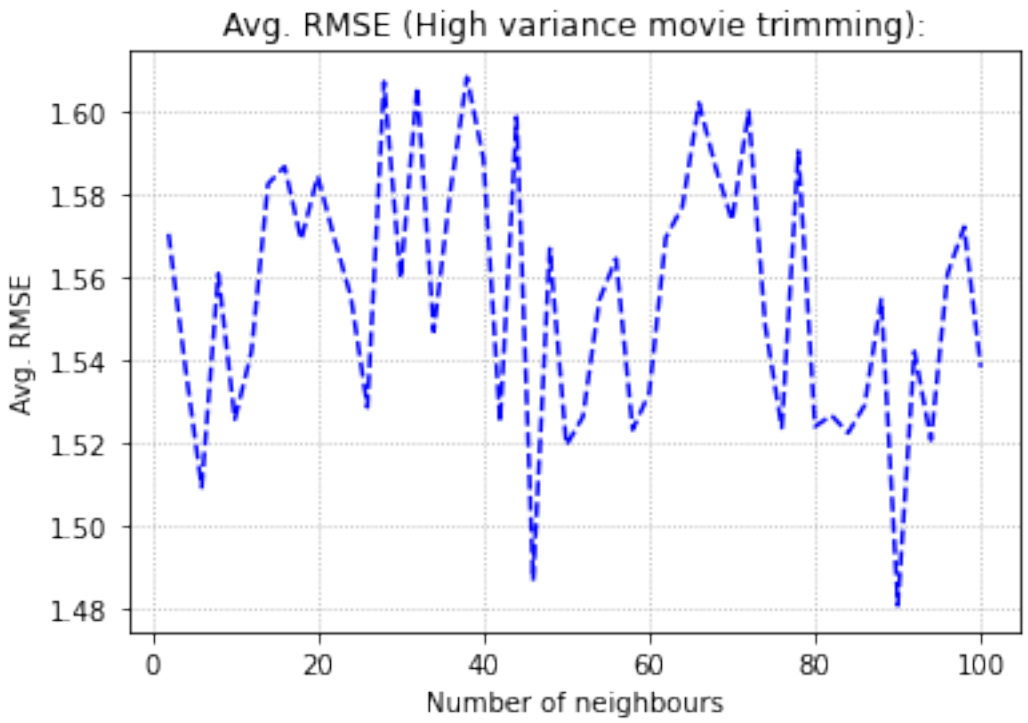
[23]:
```
rmse_var = []
kf =  KFold(n_splits=10)
dict_of_items = {}
for j in ratings_dataset.raw_ratings:
    if j[1] in dict_of_items.keys():
        dict_of_items[j[1]].append(j[2])
    else:
        dict_of_items[j[1]] = []
        dict_of_items[j[1]].append(j[2])

for item in k:
    local_rmse = []
    print('Testing for k =',item)
    for trainset, testset in kf.split(ratings_dataset):
        trimmed_set = [j for j in testset if (np.var(dict_of_items[j[1]]) >= 2
 ↪and len(dict_of_items[j[1]]) >= 5)]
        res = KNNWithMeans(k=item,sim_options={'name':'pearson'},verbose=False).
 ↪fit(trainset).test(trimmed_set)
        local_rmse.append(accuracy.rmse(res,verbose=False))
    rmse_var.append(np.mean(local_rmse))
```

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

```
Testing for k = 50
Testing for k = 52
Testing for k = 54
Testing for k = 56
Testing for k = 58
Testing for k = 60
Testing for k = 62
Testing for k = 64
Testing for k = 66
Testing for k = 68
Testing for k = 70
Testing for k = 72
Testing for k = 74
Testing for k = 76
Testing for k = 78
Testing for k = 80
Testing for k = 82
Testing for k = 84
Testing for k = 86
Testing for k = 88
Testing for k = 90
Testing for k = 92
Testing for k = 94
Testing for k = 96
Testing for k = 98
Testing for k = 100
```

[25]:
```
plt.plot(k,rmse_var,linestyle='--',color='b')
plt.grid(linestyle=':')
plt.title('Avg. RMSE (High variance movie trimming):')
plt.ylabel('Avg. RMSE')
plt.xlabel('Number of neighbours')
plt.savefig('Q14.png',dpi=300,bbox_inches='tight')
plt.show()
```



[26]:
```
print("Minimum avg. RMSE (High variance movie trimming):", min(rmse_var))
```

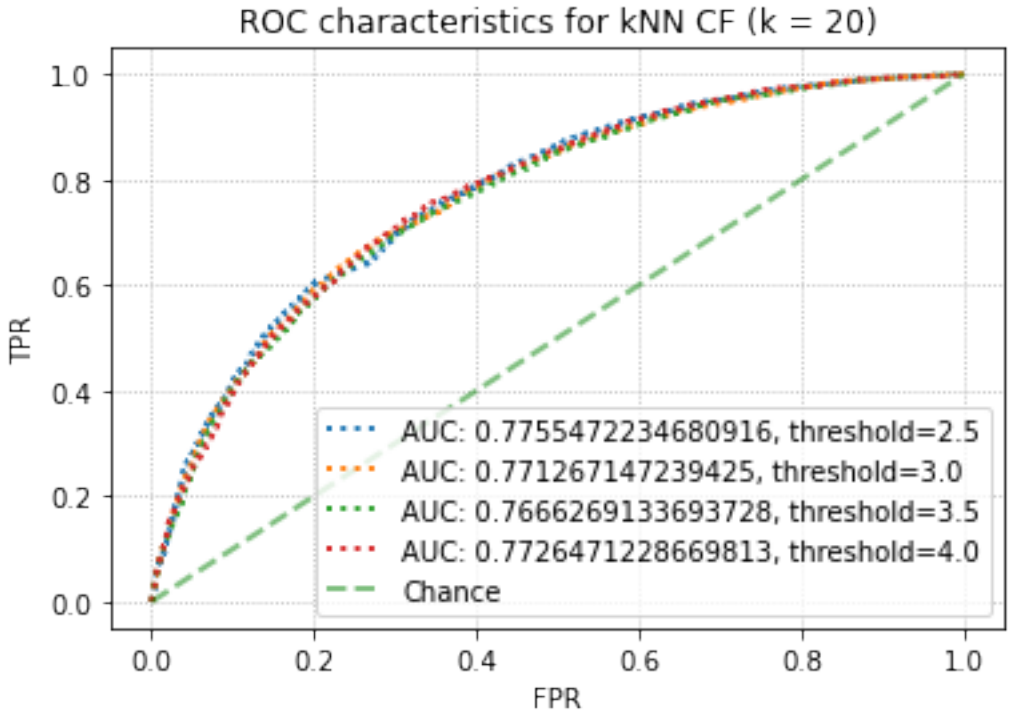
```
Minimum avg. RMSE (High variance movie trimming): 1.4807749408081747
```

8 Question 15

[40]:
```
k = 20
thres = [2.5, 3.0, 3.5, 4.0]
trainset, testset = train_test_split(ratings_dataset, test_size=0.1)
res  = KNNWithMeans(k=k,sim_options={'name':'pearson'},verbose=False).
 ↪fit(trainset).test(testset)
```

[56]:
```
fig, ax = plt.subplots()
for item in thres:
    thresholded_out = []
    for row in res:
        if row.r_ui > item:
            thresholded_out.append(1)
        else:
            thresholded_out.append(0)
    fpr, tpr, thresholds = roc_curve(thresholded_out, [row.est for row in res])
    ax.plot(fpr, tpr,lw=2,linestyle=':',label="AUC: "+str(auc(fpr,tpr))+',␣
 ↪threshold='+str(item))
```

```
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='g', label='Chance', alpha=.
↪5)
plt.legend(loc='best')
plt.grid(linestyle=':')
plt.title('ROC characteristics for kNN CF (k = 20)')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.savefig('Q15.png',dpi=300,bbox_inches='tight')
plt.show()
```

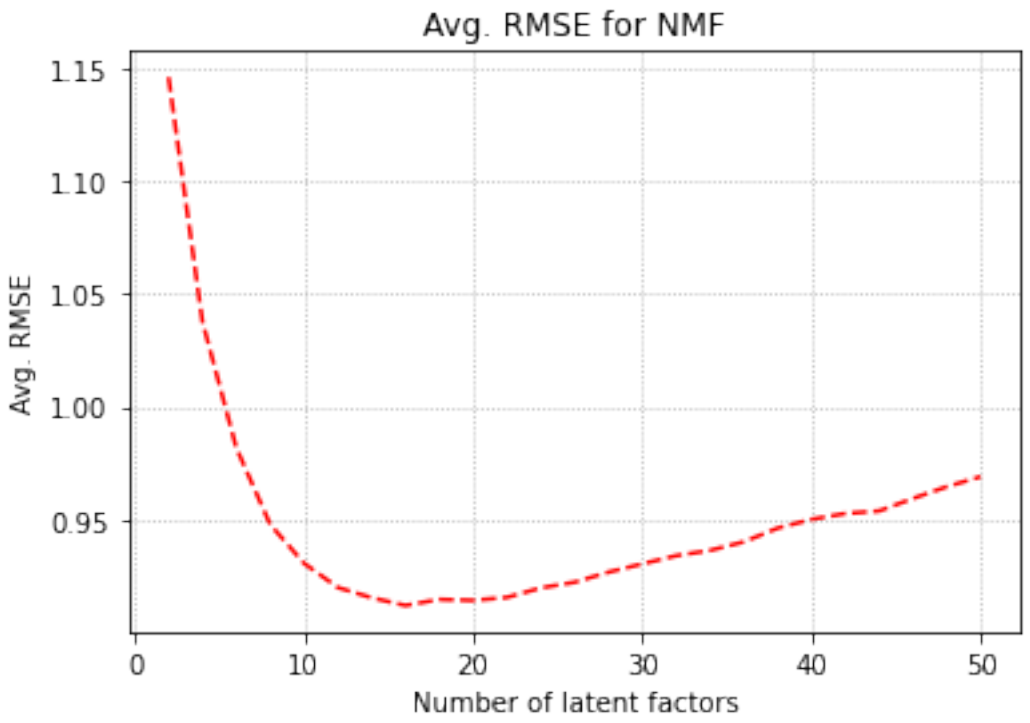


9 Question 17

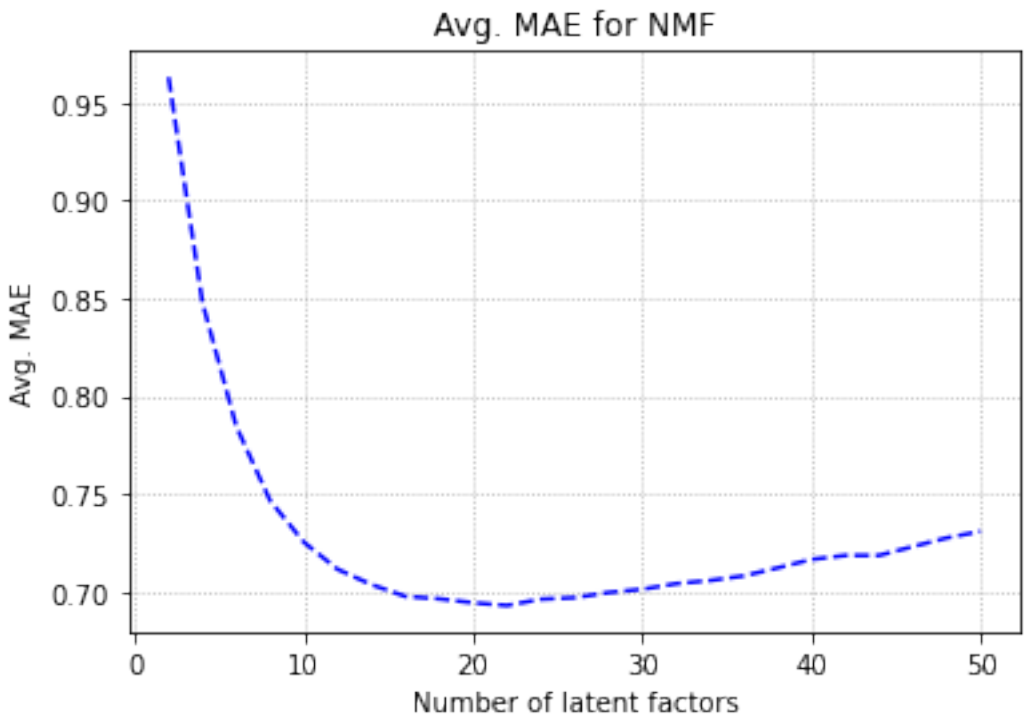
[68]:
```
k = np.arange(2,52,2)
rmse_NMF = []
mae_NMF = []
for item in k:
    print('Testing for k =',item)
    res = cross_validate(NMF(n_factors=item,n_epochs=50,verbose=False),
                         measures=['rmse','mae'],data =␣
↪ratings_dataset,cv=10,n_jobs=-1)
    rmse_NMF.append(np.mean(res['test_rmse']))
    mae_NMF.append(np.mean(res['test_mae']))
```

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

[87]:
```
plt.plot(k,rmse_NMF,linestyle='--',color='r')
plt.grid(linestyle=':')
plt.title('Avg. RMSE for NMF')
plt.ylabel('Avg. RMSE')
plt.xlabel('Number of latent factors')
plt.savefig('Q17a.png',dpi=300,bbox_inches='tight')
plt.show()
```



[88]:
```
plt.plot(k,mae_NMF,linestyle='--',color='b')
plt.grid(linestyle=':')
plt.title('Avg. MAE for NMF')
plt.ylabel('Avg. MAE')
plt.xlabel('Number of latent factors')
plt.savefig('Q17b.png',dpi=300,bbox_inches='tight')
plt.show()
```



10 Question 18

[89]:
```
print("Minimum avg. RMSE (NMF): %f, value of k: %d" % (min(rmse_NMF),k[[i for␣
↪i, x in enumerate(rmse_NMF) if x == min(rmse_NMF)][0]]))
print("Minimum avg. MAE (NMF): %f, value of k: %d" % (min(mae_NMF),k[[i for i,␣
↪x in enumerate(mae_NMF) if x == min(mae_NMF)][0]]))
```

```
Minimum avg. RMSE (NMF): 0.912293, value of k: 16
Minimum avg. MAE (NMF): 0.692898, value of k: 22
```

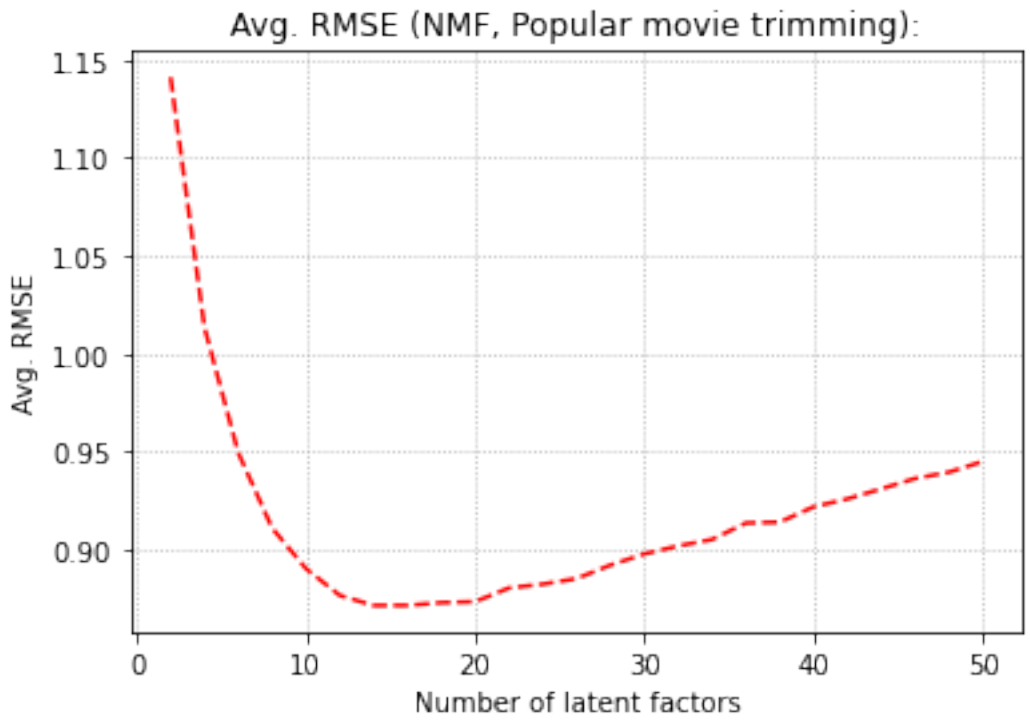
11 Question 19

[82]:
```
rmse_NMF_pop = []
kf =  KFold(n_splits=10)
for item in k:
    local_rmse = []
    print('Testing for k =',item)
    for trainset, testset in kf.split(ratings_dataset):
        trim_list = []
        unique, counts = np.unique([row[1] for row in testset],␣
↪return_counts=True)
        for i in range(len(counts)):
```

```
            if(counts[i]<=2):
                trim_list.append(unique[i])
        trimmed_set = [j for j in testset if j[1] not in trim_list]
        res = NMF(n_factors=item,n_epochs=50,verbose=False).fit(trainset).
↪test(trimmed_set)
        local_rmse.append(accuracy.rmse(res,verbose=False))
    rmse_NMF_pop.append(np.mean(local_rmse))
```

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

[85]:
```
plt.plot(k,rmse_NMF_pop,linestyle='--',color='r')
plt.grid(linestyle=':')
plt.title('Avg. RMSE (NMF, Popular movie trimming):')
plt.ylabel('Avg. RMSE')
plt.xlabel('Number of latent factors')
plt.savefig('Q19.png',dpi=300,bbox_inches='tight')
plt.show()
```



[86]:
```
print("Minimum avg. RMSE (NMF, Popular movie trimming):", min(rmse_NMF_pop))
```

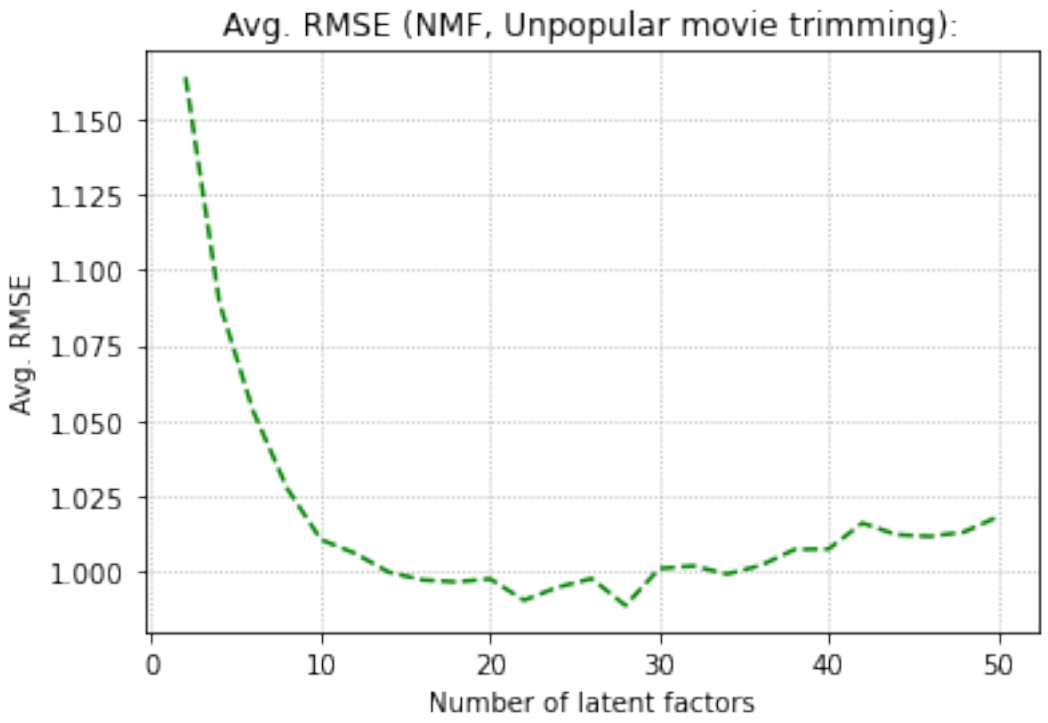
```
Minimum avg. RMSE (NMF, Popular movie trimming): 0.8716691678332561
```

12 Question 20

[90]:
```
rmse_NMF_unpop = []
kf =  KFold(n_splits=10)
for item in k:
    local_rmse = []
    print('Testing for k =',item)
    for trainset, testset in kf.split(ratings_dataset):
        trim_list = []
        unique, counts = np.unique([row[1] for row in testset],␣
↪return_counts=True)
        for i in range(len(counts)):
            if(counts[i]>2):
                trim_list.append(unique[i])
        trimmed_set = [j for j in testset if j[1] not in trim_list]
        res = NMF(n_factors=item,n_epochs=50,verbose=False).fit(trainset).
↪test(trimmed_set)
        local_rmse.append(accuracy.rmse(res,verbose=False))
    rmse_NMF_unpop.append(np.mean(local_rmse))
```

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

[91]:
```
plt.plot(k,rmse_NMF_unpop,linestyle='--',color='g')
plt.grid(linestyle=':')
plt.title('Avg. RMSE (NMF, Unpopular movie trimming):')
plt.ylabel('Avg. RMSE')
plt.xlabel('Number of latent factors')
plt.savefig('Q20.png',dpi=300,bbox_inches='tight')
plt.show()
```



[92]:
```
print("Minimum avg. RMSE (NMF, Unpopular movie trimming):", min(rmse_NMF_unpop))
```

```
Minimum avg. RMSE (NMF, Unpopular movie trimming): 0.9884940417020275
```

13 Question 21

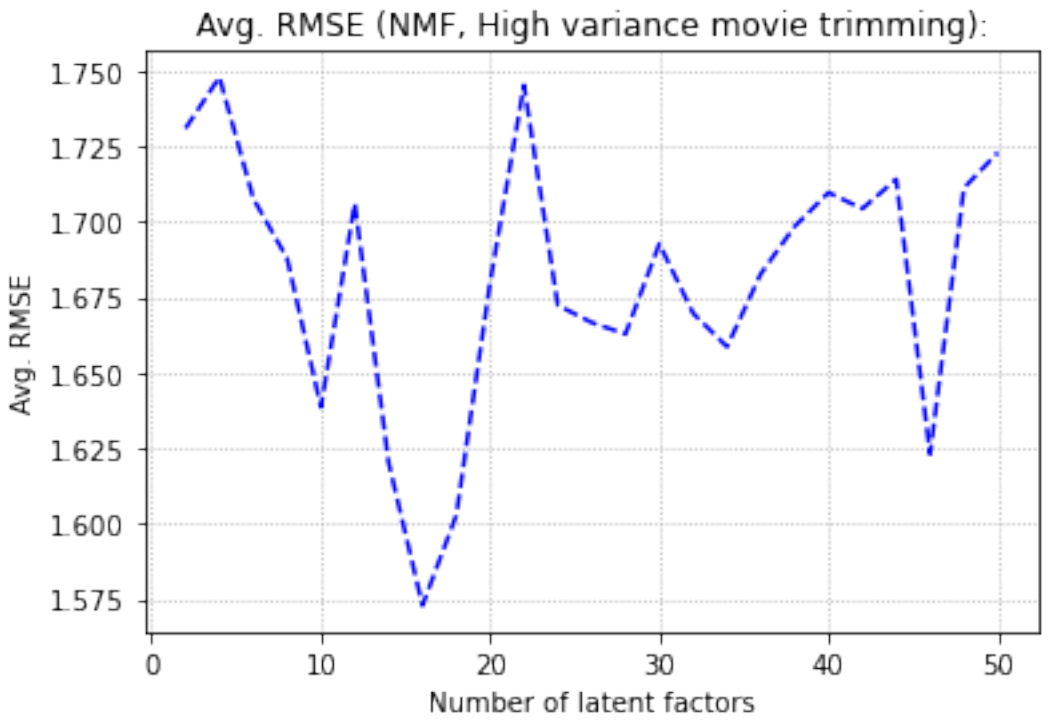
[93]:
```
rmse_NMF_var = []
kf =  KFold(n_splits=10)
dict_of_items = {}
for j in ratings_dataset.raw_ratings:
    if j[1] in dict_of_items.keys():
        dict_of_items[j[1]].append(j[2])
    else:
        dict_of_items[j[1]] = []
        dict_of_items[j[1]].append(j[2])

for item in k:
    local_rmse = []
    print('Testing for k =',item)
    for trainset, testset in kf.split(ratings_dataset):
        trimmed_set = [j for j in testset if (np.var(dict_of_items[j[1]]) >= 2␣
 ↪and len(dict_of_items[j[1]]) >= 5)]
```

```
        res = NMF(n_factors=item,n_epochs=50,verbose=False).fit(trainset).
↪test(trimmed_set)
        local_rmse.append(accuracy.rmse(res,verbose=False))
    rmse_NMF_var.append(np.mean(local_rmse))
```

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

[99]:
```
plt.plot(k,rmse_NMF_var,linestyle='--',color='b')
plt.grid(linestyle=':')
plt.title('Avg. RMSE (NMF, High variance movie trimming):')
plt.ylabel('Avg. RMSE')
plt.xlabel('Number of latent factors')
plt.savefig('Q21.png',dpi=300,bbox_inches='tight')
plt.show()
```



[95]:
```
print("Minimum avg. RMSE (NMF, High variance movie trimming):",␣
↪min(rmse_NMF_var))
```

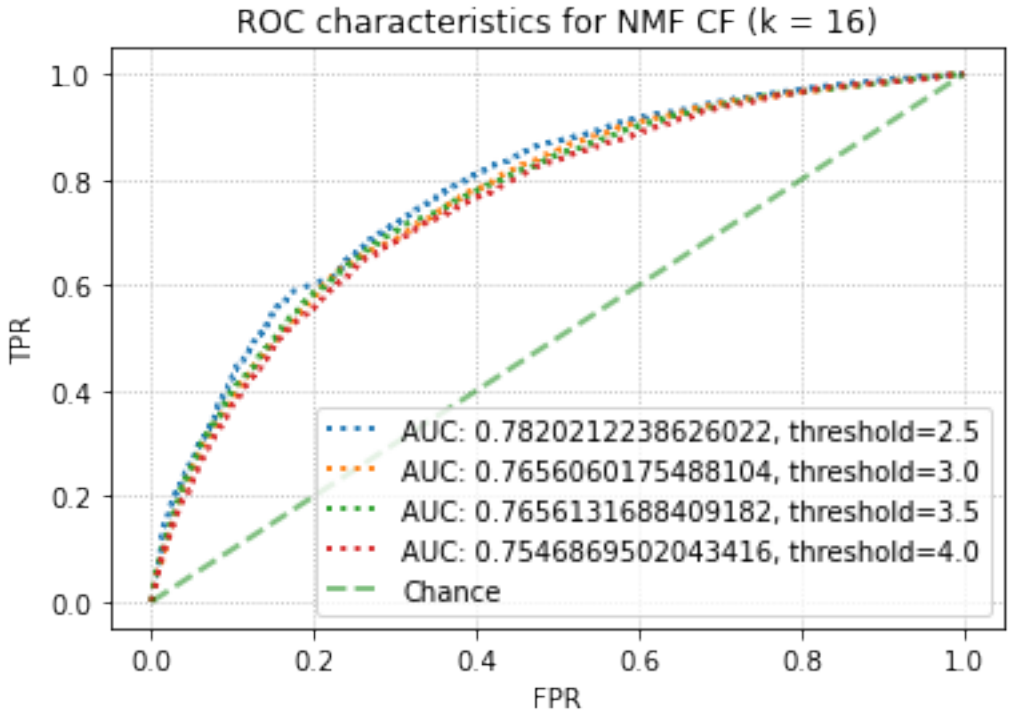
```
Minimum avg. RMSE (NMF, High variance movie trimming): 1.572969068822702
```

14 Question 22

[100]:
```
k = k[[i for i, x in enumerate(rmse_NMF) if x == min(rmse_NMF)][0]]
thres = [2.5, 3.0, 3.5, 4.0]
trainset, testset = train_test_split(ratings_dataset, test_size=0.1)
res  = NMF(n_factors=k,n_epochs=50,verbose=False).fit(trainset).test(testset)
```

[101]:
```
fig, ax = plt.subplots()
for item in thres:
    thresholded_out = []
    for row in res:
        if row.r_ui > item:
            thresholded_out.append(1)
        else:
            thresholded_out.append(0)
    fpr, tpr, thresholds = roc_curve(thresholded_out, [row.est for row in res])
    ax.plot(fpr, tpr,lw=2,linestyle=':',label="AUC: "+str(auc(fpr,tpr))+',␣
↪threshold='+str(item))
```

```
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='g', label='Chance', alpha=.
↪5)
plt.legend(loc='best')
plt.grid(linestyle=':')
plt.title('ROC characteristics for NMF CF (k = 16)')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.savefig('Q22.png',dpi=300,bbox_inches='tight')
plt.show()
```



15 Question 23

[132]:
```
genre = pd.read_csv(dataset_folder+'movies.
↪csv',usecols=['movieId','title','genres'])
trainset, testset = train_test_split(ratings_dataset, test_size=0.1)
nmf = NMF(n_factors=20,n_epochs=50,verbose=False)
nmf.fit(trainset).test(testset)
U = nmf.pu
V = nmf.qi
```

[133]:
```
cols = [1,3,5,7,11,15,19]
for item in cols:
```

```
    print('Column number of V: ',item)
    selected_col = V[:,item]
    sorted_col = np.argsort(selected_col)[::-1]
    for i in sorted_col[0:10]:
        print(genre['genres'][i])
    print('-------------------------------------------')
```

```
Column number of V:  1
Drama|Romance
Action|Adventure|Sci-Fi
Comedy|Drama|Romance
Drama
Comedy|Horror
Drama|Film-Noir|Mystery|Romance
Adventure|Drama|Western
Comedy|Drama
Comedy|Drama
Comedy|Drama
-------------------------------------------
Column number of V:  3
Comedy|Drama
Sci-Fi
Crime|Drama|Thriller
Action|Crime
Comedy|Crime|Romance
Horror|Sci-Fi
Comedy|Romance
Action|Adventure|Fantasy
Comedy
Comedy|Romance
-------------------------------------------
Column number of V:  5
Drama
Comedy
Children|Comedy
Horror|Mystery
Drama|Romance
Drama|Romance
Adventure|Children|Comedy
Action|Sci-Fi|Thriller|Western
Drama|Thriller
Comedy|Drama|Romance|Sci-Fi
-------------------------------------------
Column number of V:  7
Drama
Drama|Mystery|Sci-Fi
Action|Crime|Drama|IMAX
```

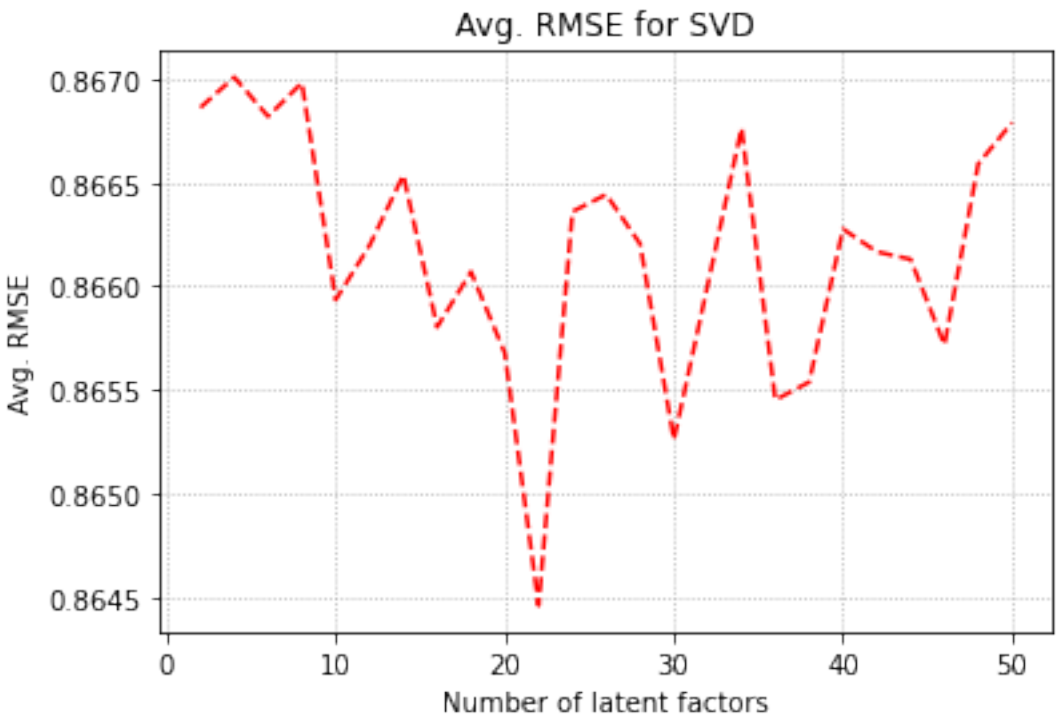
```
Comedy
Children|Drama
Horror|Thriller
Action|Crime|Drama|Thriller
Comedy
Action|Comedy
Horror
-------------------------------------------
Column number of V:  11
Drama
Comedy|Crime
Comedy
Drama
Comedy|Drama
Adventure|Drama
Comedy|Sci-Fi
Comedy
Action|Adventure|Sci-Fi
Adventure|Children|Comedy|Drama
-------------------------------------------
Column number of V:  15
Comedy
Action|Comedy
Crime|Drama|Thriller
Comedy|Drama
Action|Drama|Thriller
Comedy
Children|Comedy
Drama|Thriller
Action|Adventure|Crime|Thriller
Crime|Drama|Thriller
-------------------------------------------
Column number of V:  19
Comedy
Comedy
Documentary|Drama
Drama
Drama
Drama
Action|Adventure|Sci-Fi
Drama
Action|Horror|Sci-Fi
Adventure|Animation|Children|Comedy|Fantasy
-------------------------------------------
```

16 Question 24

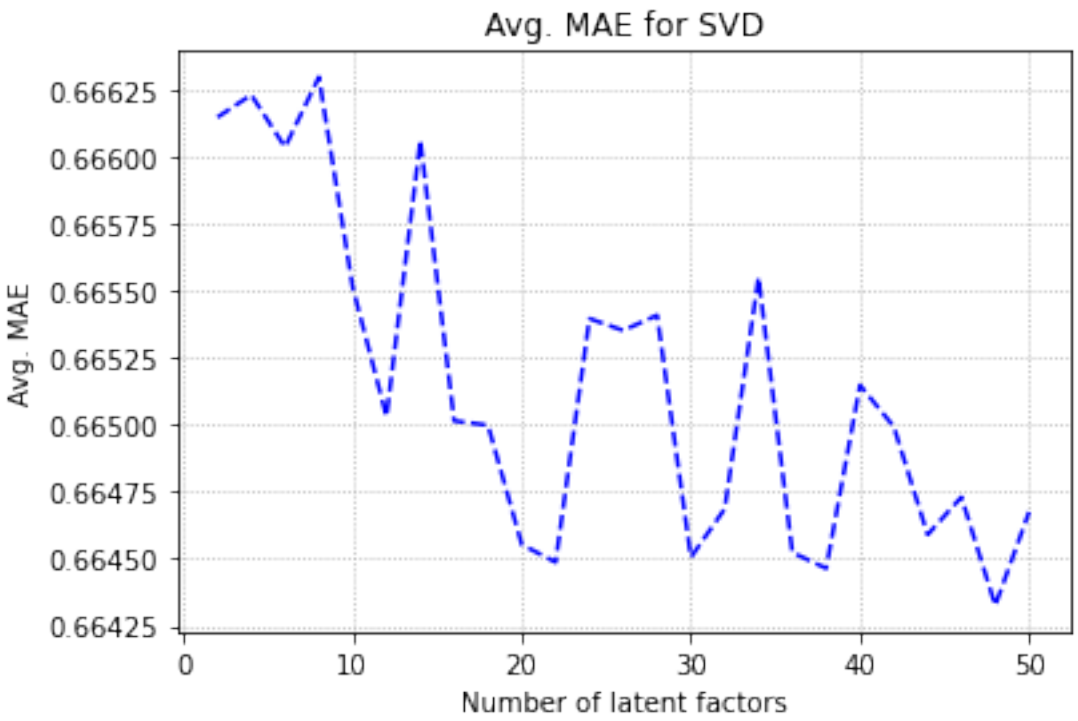
[135]:
```
k = np.arange(2,52,2)
rmse_SVD = []
mae_SVD = []
for item in k:
    print('Testing for k =',item)
    res = cross_validate(SVD(n_factors=item,n_epochs=20,verbose=False),
                         measures=['rmse','mae'],data =␣
 ↪ratings_dataset,cv=10,n_jobs=-1)
    rmse_SVD.append(np.mean(res['test_rmse']))
    mae_SVD.append(np.mean(res['test_mae']))
```

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

[136]:
```
plt.plot(k,rmse_SVD,linestyle='--',color='r')
plt.grid(linestyle=':')
plt.title('Avg. RMSE for SVD')
plt.ylabel('Avg. RMSE')
plt.xlabel('Number of latent factors')
plt.savefig('Q24a.png',dpi=300,bbox_inches='tight')
plt.show()
```



[137]:
```
plt.plot(k,mae_SVD,linestyle='--',color='b')
plt.grid(linestyle=':')
plt.title('Avg. MAE for SVD')
plt.ylabel('Avg. MAE')
plt.xlabel('Number of latent factors')
plt.savefig('Q24b.png',dpi=300,bbox_inches='tight')
plt.show()
```



17 Question 25

[138]:
```
print("Minimum avg. RMSE (SVD): %f, value of k: %d" % (min(rmse_SVD),k[[i for␣
 ↪i, x in enumerate(rmse_SVD) if x == min(rmse_SVD)][0]]))
print("Minimum avg. MAE (SVD): %f, value of k: %d" % (min(mae_SVD),k[[i for i,␣
 ↪x in enumerate(mae_SVD) if x == min(mae_SVD)][0]]))
```

```
Minimum avg. RMSE (SVD): 0.864458, value of k: 22
Minimum avg. MAE (SVD): 0.664326, value of k: 48
```

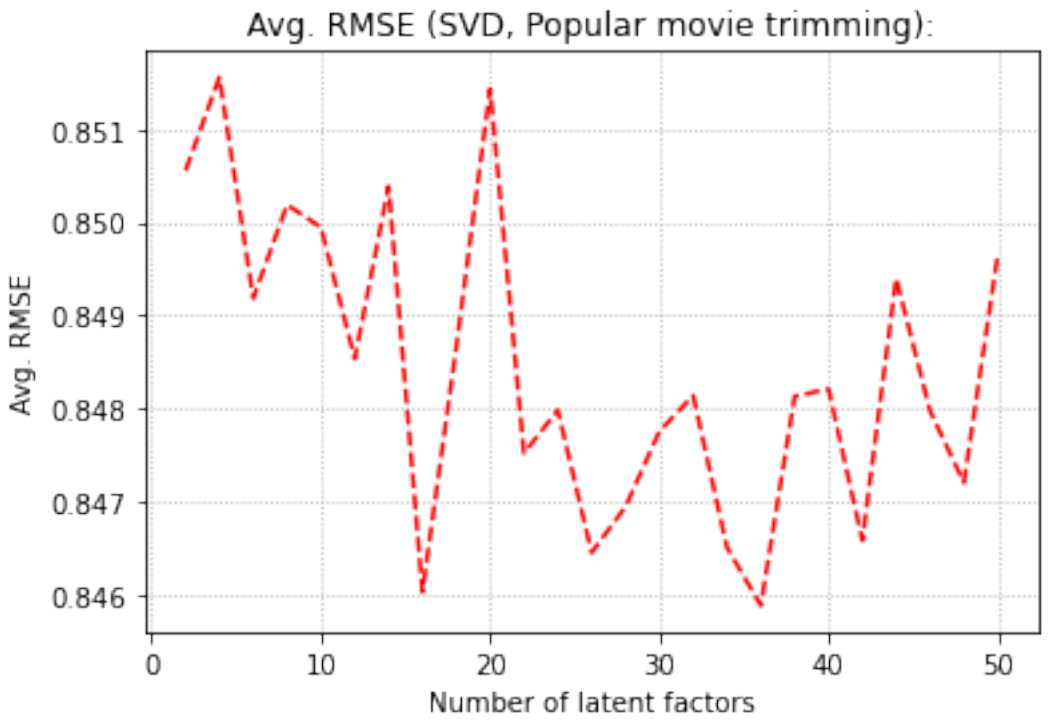
18 Question 26

[139]:
```
rmse_SVD_pop = []
kf =  KFold(n_splits=10)
for item in k:
    local_rmse = []
    print('Testing for k =',item)
    for trainset, testset in kf.split(ratings_dataset):
        trim_list = []
        unique, counts = np.unique([row[1] for row in testset],␣
 ↪return_counts=True)
        for i in range(len(counts)):
```

```
            if(counts[i]<=2):
                trim_list.append(unique[i])
        trimmed_set = [j for j in testset if j[1] not in trim_list]
        res = SVD(n_factors=item,n_epochs=20,verbose=False).fit(trainset).
↪test(trimmed_set)
        local_rmse.append(accuracy.rmse(res,verbose=False))
    rmse_SVD_pop.append(np.mean(local_rmse))
```

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

[140]:
```
plt.plot(k,rmse_SVD_pop,linestyle='--',color='r')
plt.grid(linestyle=':')
plt.title('Avg. RMSE (SVD, Popular movie trimming):')
plt.ylabel('Avg. RMSE')
plt.xlabel('Number of latent factors')
plt.savefig('Q26.png',dpi=300,bbox_inches='tight')
plt.show()
```



[141]:
```
print("Minimum avg. RMSE (SVD, Popular movie trimming):", min(rmse_SVD_pop))
```

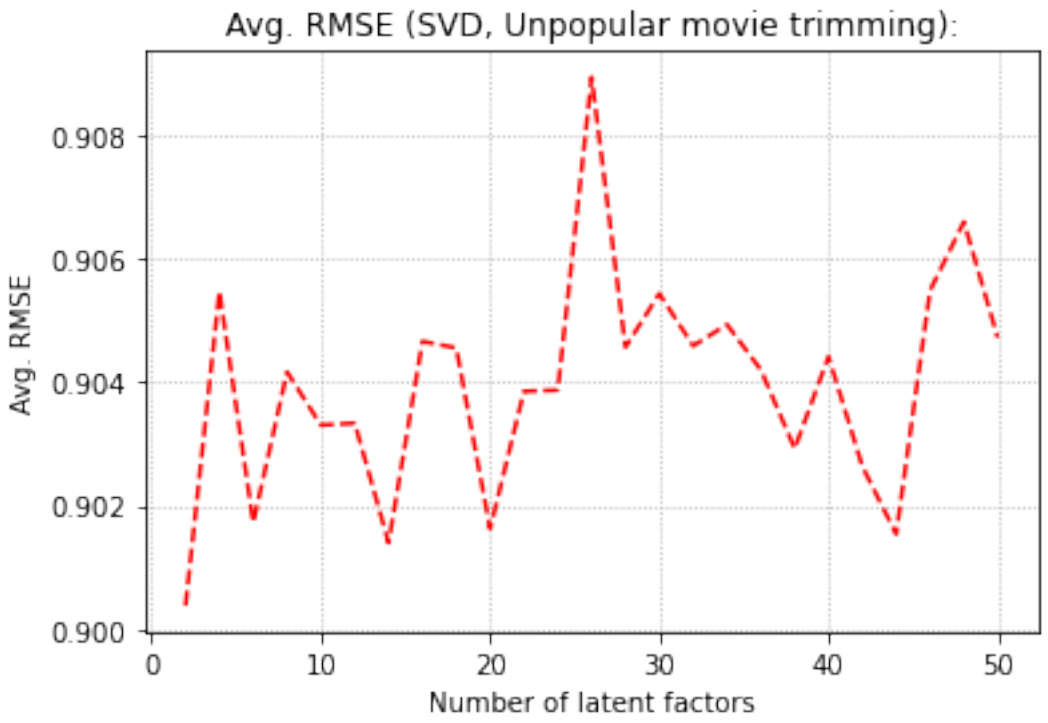
```
Minimum avg. RMSE (SVD, Popular movie trimming): 0.8458905703396464
```

19 Question 27

[142]:
```
rmse_SVD_unpop = []
kf =  KFold(n_splits=10)
for item in k:
    local_rmse = []
    print('Testing for k =',item)
    for trainset, testset in kf.split(ratings_dataset):
        trim_list = []
        unique, counts = np.unique([row[1] for row in testset],␣
↪return_counts=True)
        for i in range(len(counts)):
            if(counts[i]>2):
                trim_list.append(unique[i])
        trimmed_set = [j for j in testset if j[1] not in trim_list]
        res = SVD(n_factors=item,n_epochs=20,verbose=False).fit(trainset).
↪test(trimmed_set)
        local_rmse.append(accuracy.rmse(res,verbose=False))
    rmse_SVD_unpop.append(np.mean(local_rmse))
```

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

[143]:
```
plt.plot(k,rmse_SVD_unpop,linestyle='--',color='r')
plt.grid(linestyle=':')
plt.title('Avg. RMSE (SVD, Unpopular movie trimming):')
plt.ylabel('Avg. RMSE')
plt.xlabel('Number of latent factors')
plt.savefig('Q27.png',dpi=300,bbox_inches='tight')
plt.show()
```



[144]:
```
print("Minimum avg. RMSE (SVD, Unpopular movie trimming):", min(rmse_SVD_unpop))
```

```
Minimum avg. RMSE (SVD, Unpopular movie trimming): 0.900390027866829
```

20 Question 28

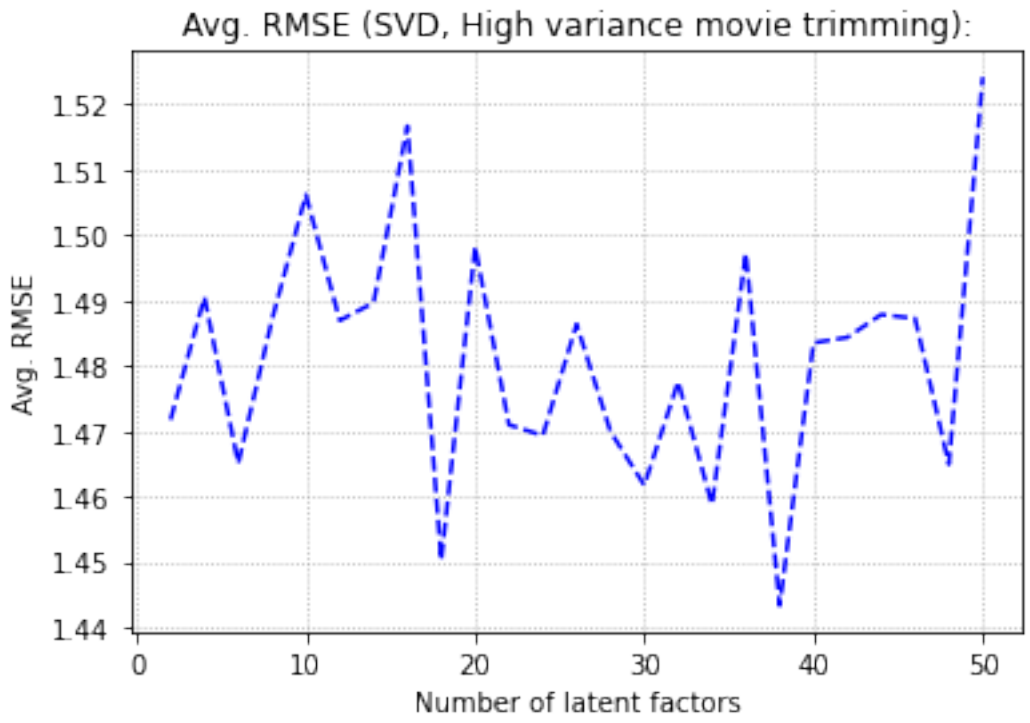
[145]:
```
rmse_SVD_var = []
kf =  KFold(n_splits=10)
dict_of_items = {}
for j in ratings_dataset.raw_ratings:
    if j[1] in dict_of_items.keys():
        dict_of_items[j[1]].append(j[2])
    else:
        dict_of_items[j[1]] = []
        dict_of_items[j[1]].append(j[2])

for item in k:
    local_rmse = []
    print('Testing for k =',item)
    for trainset, testset in kf.split(ratings_dataset):
        trimmed_set = [j for j in testset if (np.var(dict_of_items[j[1]]) >= 2␣
 ↪and len(dict_of_items[j[1]]) >= 5)]
```

```
        res = SVD(n_factors=item,n_epochs=20,verbose=False).fit(trainset).
↪test(trimmed_set)
        local_rmse.append(accuracy.rmse(res,verbose=False))
    rmse_SVD_var.append(np.mean(local_rmse))
```

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

[146]:
```
plt.plot(k,rmse_SVD_var,linestyle='--',color='b')
plt.grid(linestyle=':')
plt.title('Avg. RMSE (SVD, High variance movie trimming):')
plt.ylabel('Avg. RMSE')
plt.xlabel('Number of latent factors')
plt.savefig('Q28.png',dpi=300,bbox_inches='tight')
plt.show()
```



[147]:
```
print("Minimum avg. RMSE (SVD, High variance movie trimming):",␣
↪min(rmse_SVD_var))
```

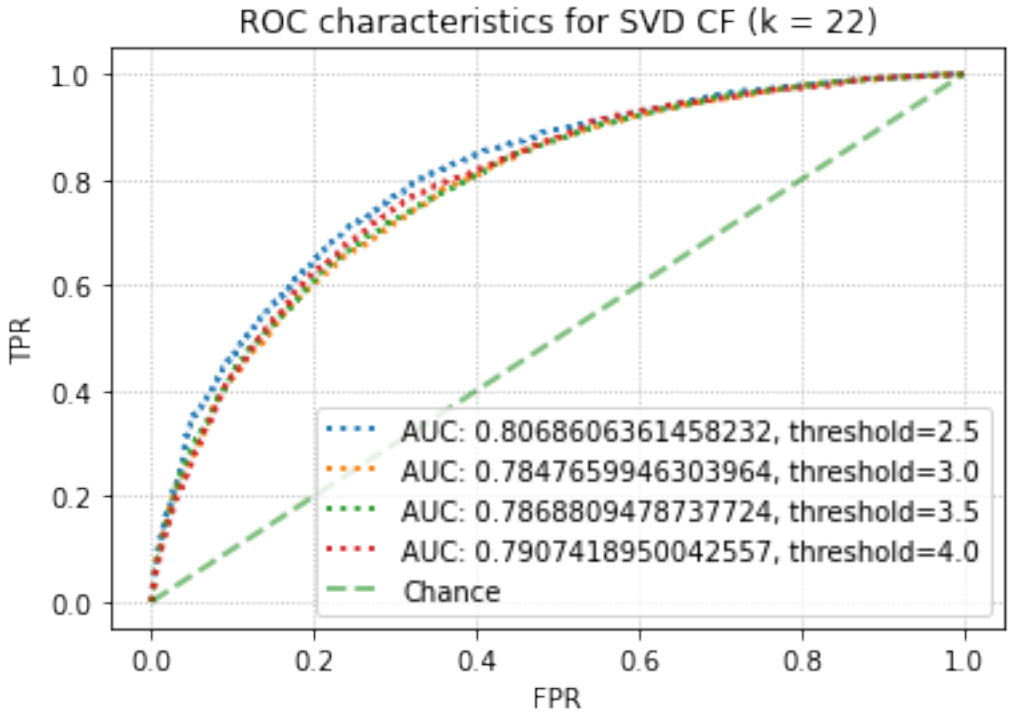
```
Minimum avg. RMSE (SVD, High variance movie trimming): 1.4432989104828147
```

21 Question 29

[148]:
```
k = k[[i for i, x in enumerate(rmse_SVD) if x == min(rmse_SVD)][0]]
thres = [2.5, 3.0, 3.5, 4.0]
trainset, testset = train_test_split(ratings_dataset, test_size=0.1)
res  = SVD(n_factors=k,n_epochs=20,verbose=False).fit(trainset).test(testset)
```

[150]:
```
fig, ax = plt.subplots()
for item in thres:
    thresholded_out = []
    for row in res:
        if row.r_ui > item:
            thresholded_out.append(1)
        else:
            thresholded_out.append(0)
    fpr, tpr, thresholds = roc_curve(thresholded_out, [row.est for row in res])
    ax.plot(fpr, tpr,lw=2,linestyle=':',label="AUC: "+str(auc(fpr,tpr))+',␣
↪threshold='+str(item))
```

```
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='g', label='Chance', alpha=.
↪5)
plt.legend(loc='best')
plt.grid(linestyle=':')
plt.title('ROC characteristics for SVD CF (k = '+ str(k)+')')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.savefig('Q29.png',dpi=300,bbox_inches='tight')
plt.show()
```



22 Question 30

[224]:
```
user_ID_set = list(set(user_ID))
mean_ratings = []
for user in user_ID_set:
    idx = np.where(user_ID == user)
    mean_ratings.append(np.mean(rating[idx]))
```

[228]:
```
kf = KFold(n_splits=10)
local_rmse = []
for trainset, testset in kf.split(ratings_dataset):
    res = [mean_ratings[int(row[0])-1] for row in testset]
```

```
    gt = [row[2] for row in testset]
    local_rmse.append(mean_squared_error(gt,res,squared=False))
rmse_naive = np.mean(local_rmse)
```

[229]:
```
print('Avg. RMSE for Naive Filtering: ',rmse_naive)
```

```
Avg. RMSE for Naive Filtering:  0.9346899216061322
```

23 Question 31

[257]:
```
local_rmse_naive_pop = []
kf =  KFold(n_splits=10)
for trainset, testset in kf.split(ratings_dataset):
    trim_list = []
    unique, counts = np.unique([row[1] for row in testset], return_counts=True)
    for i in range(len(counts)):
        if(counts[i]<=2):
            trim_list.append(unique[i])
    trimmed_set = [j for j in testset if j[1] not in trim_list]
    res = [mean_ratings[int(row[0])-1] for row in trimmed_set]
    gt = [row[2] for row in trimmed_set]
    local_rmse_naive_pop.append(mean_squared_error(gt,res,squared=False))
rmse_naive_pop = np.mean(local_rmse_naive_pop)
```

[258]:
```
print('Avg. RMSE for Naive Filtering (Popular movie trimming): ',rmse_naive_pop)
```

```
Avg. RMSE for Naive Filtering (Popular movie trimming):  0.9241698952751272
```

24 Question 32

[261]:
```
local_rmse_naive_unpop = []
kf =  KFold(n_splits=10)
for trainset, testset in kf.split(ratings_dataset):
    trim_list = []
    unique, counts = np.unique([row[1] for row in testset], return_counts=True)
    for i in range(len(counts)):
        if(counts[i]>2):
            trim_list.append(unique[i])
    trimmed_set = [j for j in testset if j[1] not in trim_list]
    res = [mean_ratings[int(row[0])-1] for row in trimmed_set]
    gt = [row[2] for row in trimmed_set]
    local_rmse_naive_unpop.append(mean_squared_error(gt,res,squared=False))
rmse_naive_unpop = np.mean(local_rmse_naive_unpop)
```

[262]:
```
print('Avg. RMSE for Naive Filtering (Unpopular movie trimming):␣
↪',rmse_naive_unpop)
```

```
Avg. RMSE for Naive Filtering (Unpopular movie trimming):  0.9545871807615829
```

25 Question 33

[263]:
```
local_rmse_naive_var = []
kf =  KFold(n_splits=10)
dict_of_items = {}
for j in ratings_dataset.raw_ratings:
    if j[1] in dict_of_items.keys():
        dict_of_items[j[1]].append(j[2])
    else:
        dict_of_items[j[1]] = []
        dict_of_items[j[1]].append(j[2])

for trainset, testset in kf.split(ratings_dataset):
    trimmed_set = [j for j in testset if (np.var(dict_of_items[j[1]]) >= 2 and␣
 ↪len(dict_of_items[j[1]]) >= 5)]
    res = [mean_ratings[int(row[0])-1] for row in trimmed_set]
    gt = [row[2] for row in trimmed_set]
    local_rmse_naive_var.append(mean_squared_error(gt,res,squared=False))
rmse_naive_var = np.mean(local_rmse_naive_var)
```

[264]:
```
print('Avg. RMSE for Naive Filtering (Unpopular movie trimming):␣
 ↪',rmse_naive_var)
```

```
Avg. RMSE for Naive Filtering (Unpopular movie trimming):  1.466818523657904
```

26 Question 34

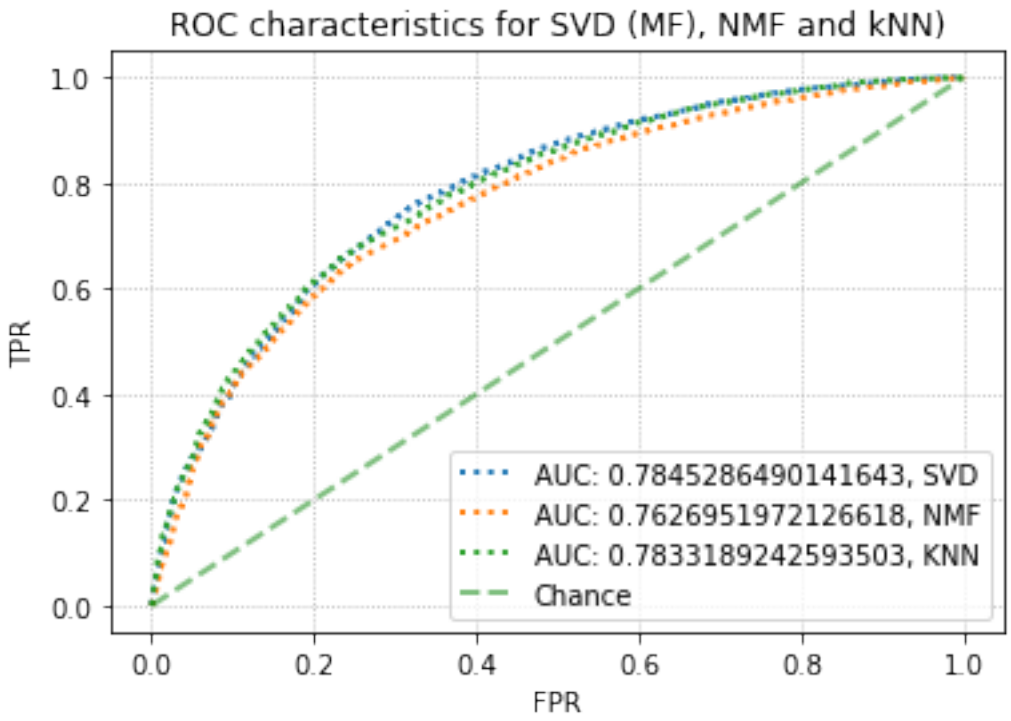
[265]:
```
trainset, testset = train_test_split(ratings_dataset, test_size=0.1)
res_SVD  = SVD(n_factors=22,n_epochs=20,verbose=False).fit(trainset).
 ↪test(testset)
res_NMF  = NMF(n_factors=16,n_epochs=50,verbose=False).fit(trainset).
 ↪test(testset)
res_KNN = KNNWithMeans(k=20,sim_options={'name':'pearson'},verbose=False).
 ↪fit(trainset).test(testset)
```

[268]:
```
fig, ax = plt.subplots()
thresholded_out = []
for row in res_SVD:
    if row.r_ui > 3:
        thresholded_out.append(1)
    else:
        thresholded_out.append(0)
fpr, tpr, thresholds = roc_curve(thresholded_out, [row.est for row in res_SVD])
ax.plot(fpr, tpr,lw=2,linestyle=':',label="AUC: "+str(auc(fpr,tpr))+', SVD')
```

```
thresholded_out = []
for row in res_NMF:
    if row.r_ui > 3:
        thresholded_out.append(1)
    else:
        thresholded_out.append(0)
fpr, tpr, thresholds = roc_curve(thresholded_out, [row.est for row in res_NMF])
ax.plot(fpr, tpr,lw=2,linestyle=':',label="AUC: "+str(auc(fpr,tpr))+', NMF')

thresholded_out = []
for row in res_KNN:
    if row.r_ui > 3:
        thresholded_out.append(1)
    else:
        thresholded_out.append(0)
fpr, tpr, thresholds = roc_curve(thresholded_out, [row.est for row in res_KNN])
ax.plot(fpr, tpr,lw=2,linestyle=':',label="AUC: "+str(auc(fpr,tpr))+', KNN')

ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='g', label='Chance', alpha=.
 ↪5)
plt.legend(loc='best')
plt.grid(linestyle=':')
plt.title('ROC characteristics for SVD (MF), NMF and kNN)')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.savefig('Q34.png',dpi=300,bbox_inches='tight')
plt.show()
```



27 Question 36

[303]:
```
t = np.arange(1,26,1)
kf = KFold(n_splits=10)
```

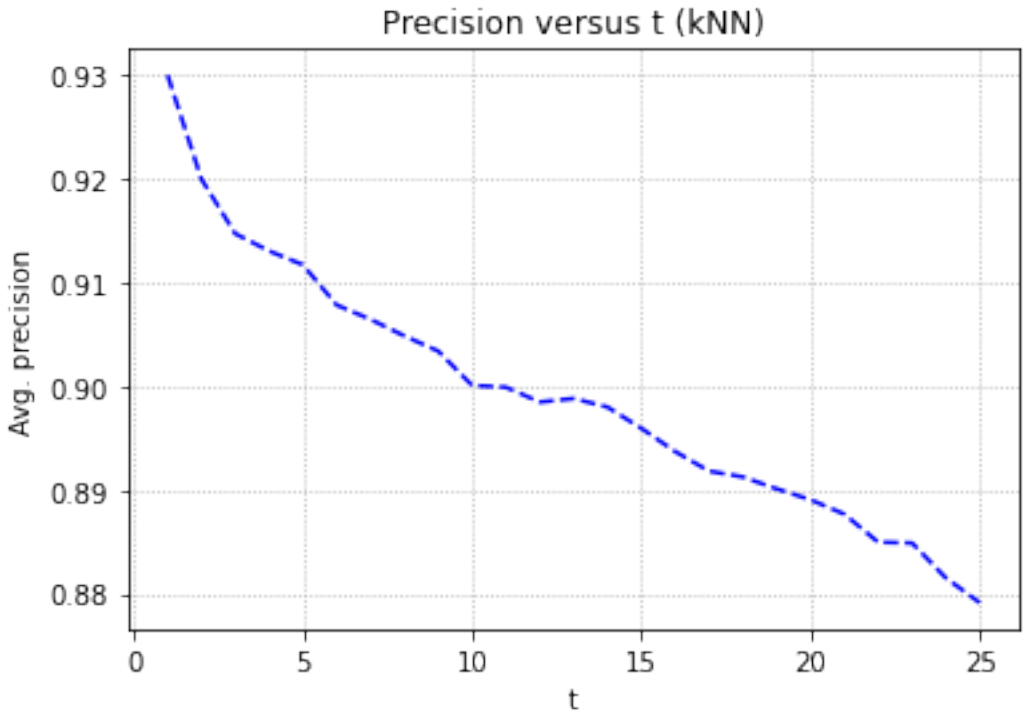
[304]:
```
prec_list_knn = []
rec_list_knn = []
for val in t:
    print('Testing for t =',val)
    precision_set = []
    recall_set = []
    for trainset, testset in kf.split(ratings_dataset):
        G = {} #dictionary of movies liked by users
        for row in testset:
            if row[0] in G.keys():
                if row[2] >= 3.0:
                    G[row[0]].add(row[1])
            else:
                G[row[0]] = set()
                if row[2] >= 3.0:
                    G[row[0]].add(row[1])
        dict_of_items = {} #dictionary of all movies rated by users
```

```
        for row in testset:
            if row[0] in dict_of_items.keys():
                dict_of_items[row[0]].append(row[1])
            else:
                dict_of_items[row[0]] = []
                dict_of_items[row[0]].append(row[1])
        mod_testset = [row for row in testset if (len(dict_of_items[row[0]]) >=␣
↪val and len(G[row[0]]) > 0)]
        res = KNNWithMeans(k=20,sim_options={'name':'pearson'},verbose=False).
↪fit(trainset).test(mod_testset)
        est_rat = {} #dictionary of estimated ratings by users
        for row in res:
            if row[0] in est_rat.keys():
                est_rat[row[0]].append((row[1],row[3]))
            else:
                est_rat[row[0]] = []
                est_rat[row[0]].append((row[1],row[3]))
        precision_u = []
        recall_u = []
        for item in est_rat.keys():
            S_all = est_rat[item]
            S_all = sorted(S_all,key=lambda x:x[1],reverse=True)
            S_t = set([row[0] for row in S_all[0:val]])
            precision_u.append(len(S_t.intersection(G[item]))/float(len(S_t)))
            recall_u.append(len(S_t.intersection(G[item]))/float(len(G[item])))
        precision_set.append(np.mean(precision_u))
        recall_set.append(np.mean(recall_u))
    prec_list_knn.append(np.mean(precision_set))
    rec_list_knn.append(np.mean(recall_set))
```

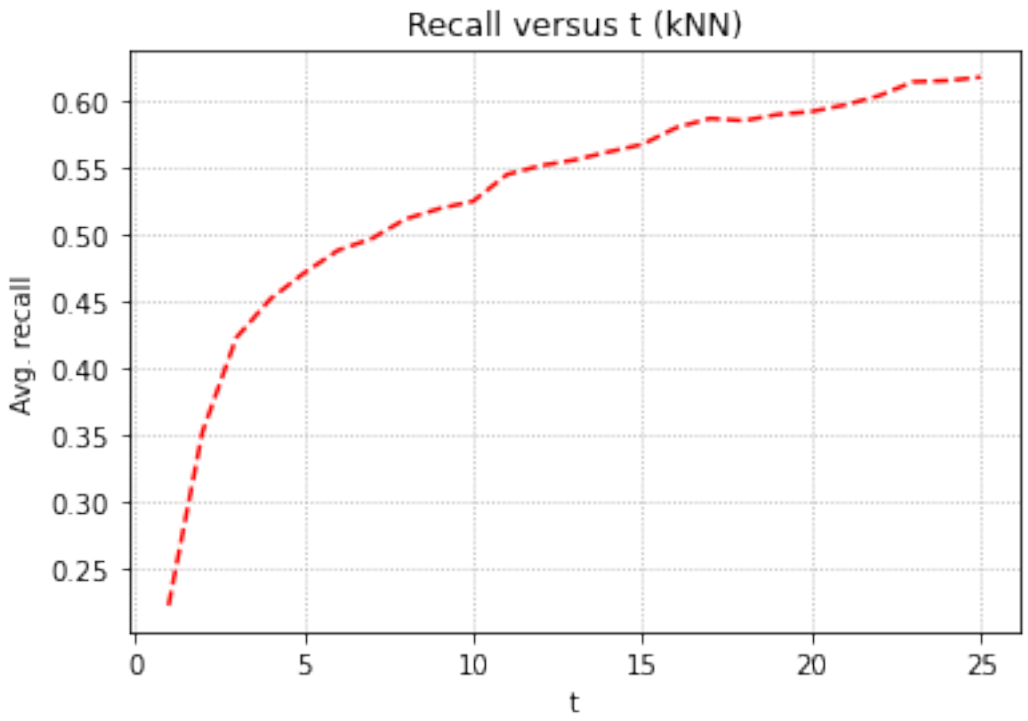
```
Testing for t = 1
Testing for t = 2
Testing for t = 3
Testing for t = 4
Testing for t = 5
Testing for t = 6
Testing for t = 7
Testing for t = 8
Testing for t = 9
Testing for t = 10
Testing for t = 11
Testing for t = 12
Testing for t = 13
Testing for t = 14
Testing for t = 15
Testing for t = 16
Testing for t = 17
```

```
Testing for t = 18
Testing for t = 19
Testing for t = 20
Testing for t = 21
Testing for t = 22
Testing for t = 23
Testing for t = 24
Testing for t = 25
```

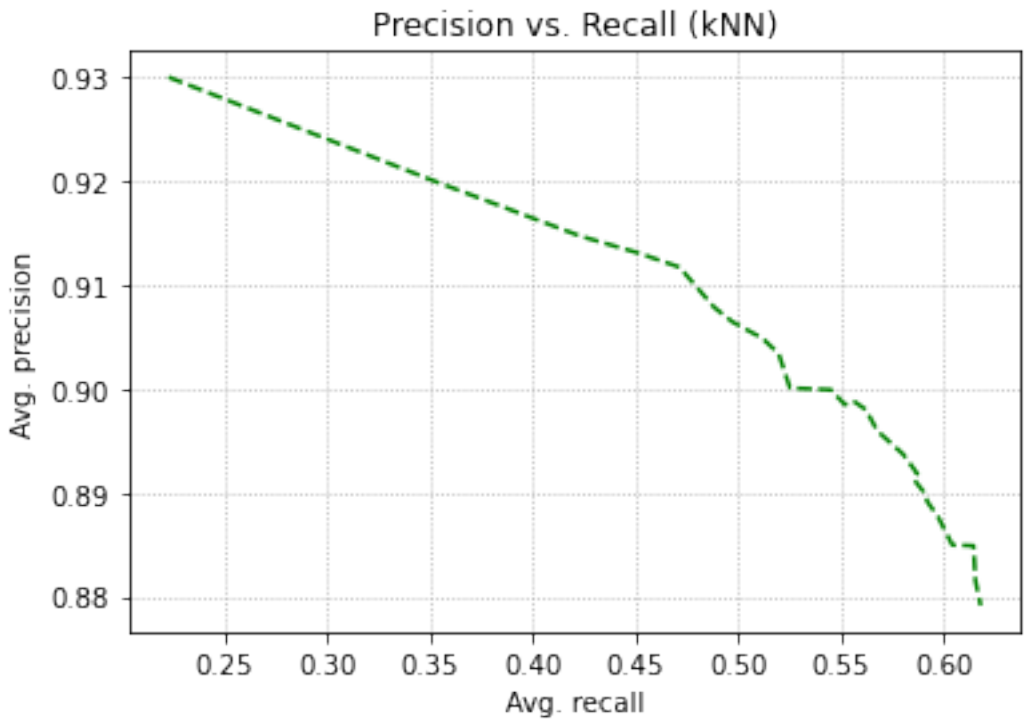
[305]:
```
plt.plot(t,prec_list_knn,linestyle='--',color='b')
plt.grid(linestyle=':')
plt.title('Precision versus t (kNN)')
plt.ylabel('Avg. precision')
plt.xlabel('t')
plt.savefig('Q36a.png',dpi=300,bbox_inches='tight')
plt.show()
```



[306]:
```
plt.plot(t,rec_list_knn,linestyle='--',color='r')
plt.grid(linestyle=':')
plt.title('Recall versus t (kNN)')
plt.ylabel('Avg. recall')
plt.xlabel('t')
plt.savefig('Q36b.png',dpi=300,bbox_inches='tight')
plt.show()
```



[308]:
```
plt.plot(rec_list_knn,prec_list_knn,linestyle='--',color='g')
plt.grid(linestyle=':')
plt.title('Precision vs. Recall (kNN)')
plt.ylabel('Avg. precision')
plt.xlabel('Avg. recall')
plt.savefig('Q36c.png',dpi=300,bbox_inches='tight')
plt.show()
```



27.1 Question 37

[310]:
```
t = np.arange(1,26,1)
kf = KFold(n_splits=10)
```

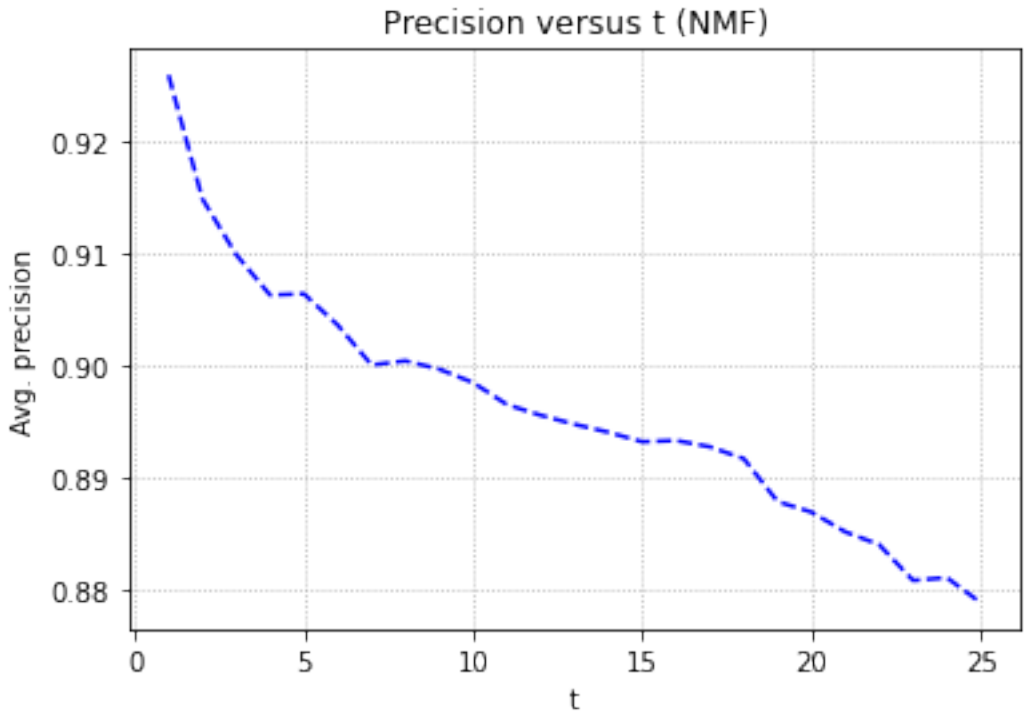
[311]:
```
prec_list_nmf = []
rec_list_nmf = []
for val in t:
    print('Testing for t =',val)
    precision_set = []
    recall_set = []
    for trainset, testset in kf.split(ratings_dataset):
        G = {} #dictionary of movies liked by users
        for row in testset:
            if row[0] in G.keys():
                if row[2] >= 3.0:
                    G[row[0]].add(row[1])
            else:
                G[row[0]] = set()
                if row[2] >= 3.0:
                    G[row[0]].add(row[1])
        dict_of_items = {} #dictionary of all movies rated by users
        for row in testset:
```

```
            if row[0] in dict_of_items.keys():
                dict_of_items[row[0]].append(row[1])
            else:
                dict_of_items[row[0]] = []
                dict_of_items[row[0]].append(row[1])
        mod_testset = [row for row in testset if (len(dict_of_items[row[0]]) >=␣
↪val and len(G[row[0]]) > 0)]
        res = NMF(n_factors=16,n_epochs=50,verbose=False).fit(trainset).
↪test(mod_testset)
        est_rat = {} #dictionary of estimated ratings by users
        for row in res:
            if row[0] in est_rat.keys():
                est_rat[row[0]].append((row[1],row[3]))
            else:
                est_rat[row[0]] = []
                est_rat[row[0]].append((row[1],row[3]))
        precision_u = []
        recall_u = []
        for item in est_rat.keys():
            S_all = est_rat[item]
            S_all = sorted(S_all,key=lambda x:x[1],reverse=True)
            S_t = set([row[0] for row in S_all[0:val]])
            precision_u.append(len(S_t.intersection(G[item]))/float(len(S_t)))
            recall_u.append(len(S_t.intersection(G[item]))/float(len(G[item])))
        precision_set.append(np.mean(precision_u))
        recall_set.append(np.mean(recall_u))
    prec_list_nmf.append(np.mean(precision_set))
    rec_list_nmf.append(np.mean(recall_set))
```

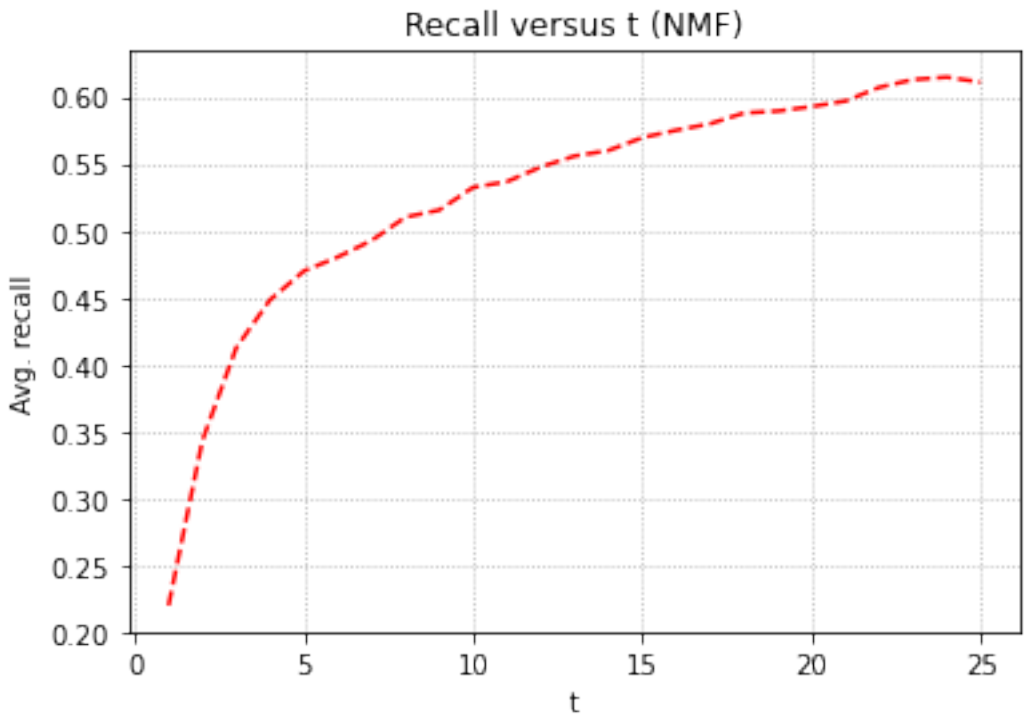
```
Testing for t = 1
Testing for t = 2
Testing for t = 3
Testing for t = 4
Testing for t = 5
Testing for t = 6
Testing for t = 7
Testing for t = 8
Testing for t = 9
Testing for t = 10
Testing for t = 11
Testing for t = 12
Testing for t = 13
Testing for t = 14
Testing for t = 15
Testing for t = 16
Testing for t = 17
Testing for t = 18
```

```
Testing for t = 19
Testing for t = 20
Testing for t = 21
Testing for t = 22
Testing for t = 23
Testing for t = 24
Testing for t = 25
```

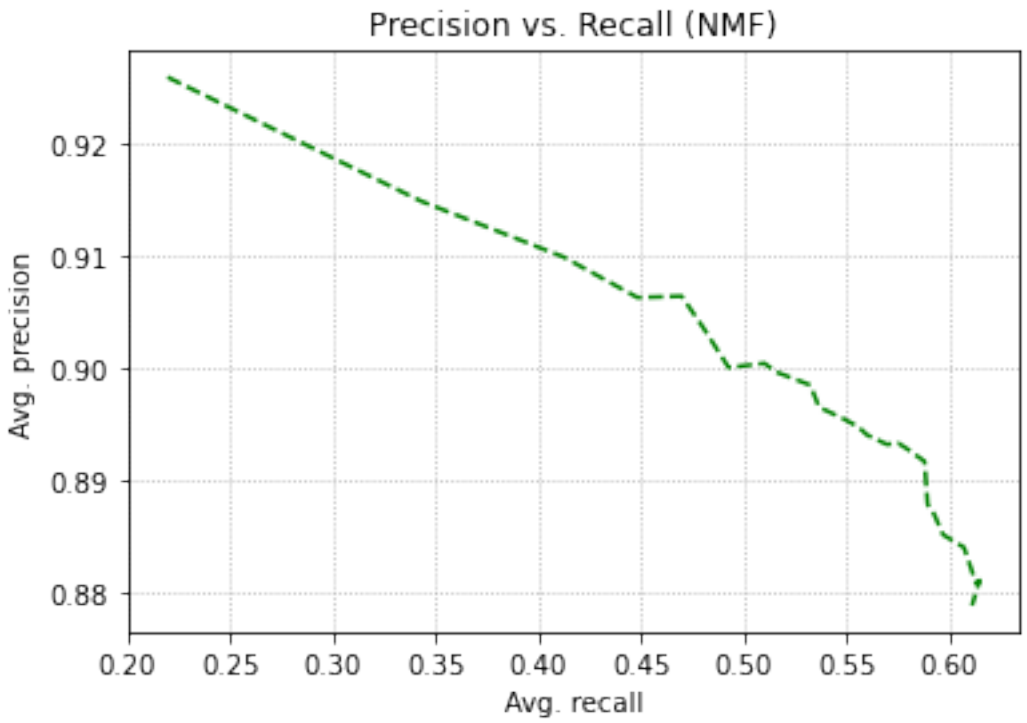
[312]:
```
plt.plot(t,prec_list_nmf,linestyle='--',color='b')
plt.grid(linestyle=':')
plt.title('Precision versus t (NMF)')
plt.ylabel('Avg. precision')
plt.xlabel('t')
plt.savefig('Q37a.png',dpi=300,bbox_inches='tight')
plt.show()
```



[313]:
```
plt.plot(t,rec_list_nmf,linestyle='--',color='r')
plt.grid(linestyle=':')
plt.title('Recall versus t (NMF)')
plt.ylabel('Avg. recall')
plt.xlabel('t')
plt.savefig('Q37b.png',dpi=300,bbox_inches='tight')
plt.show()
```



[314]:
```
plt.plot(rec_list_nmf,prec_list_nmf,linestyle='--',color='g')
plt.grid(linestyle=':')
plt.title('Precision vs. Recall (NMF)')
plt.ylabel('Avg. precision')
plt.xlabel('Avg. recall')
plt.savefig('Q37c.png',dpi=300,bbox_inches='tight')
plt.show()
```



28 Question 38

[315]:
```
t = np.arange(1,26,1)
kf = KFold(n_splits=10)
```

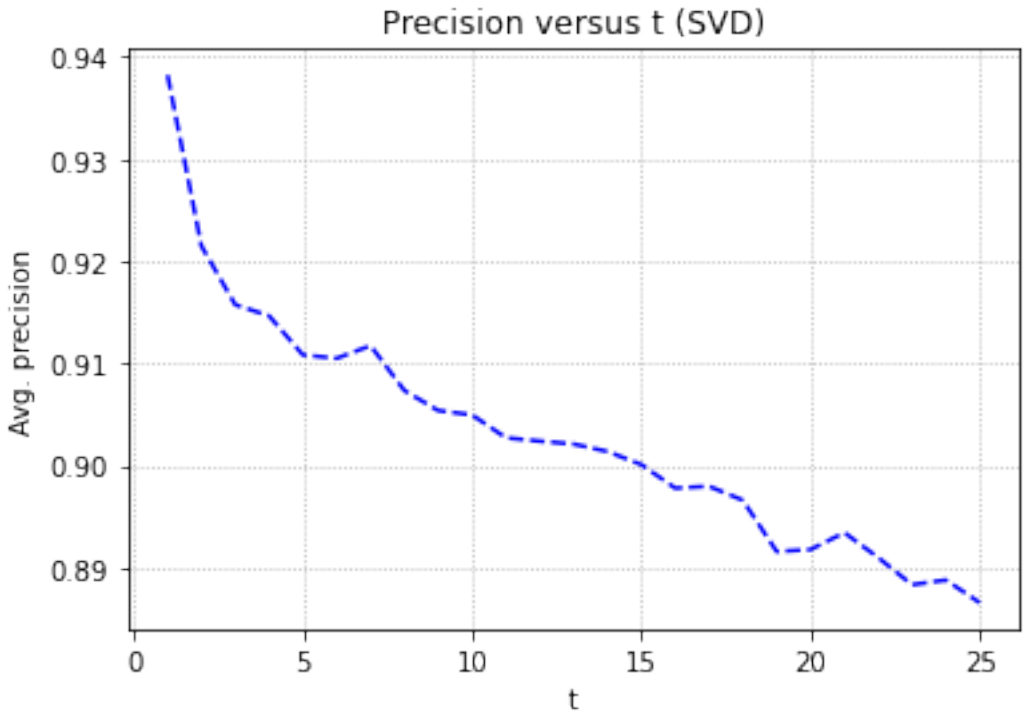
[316]:
```
prec_list_svd = []
rec_list_svd = []
for val in t:
    print('Testing for t =',val)
    precision_set = []
    recall_set = []
    for trainset, testset in kf.split(ratings_dataset):
        G = {} #dictionary of movies liked by users
        for row in testset:
            if row[0] in G.keys():
                if row[2] >= 3.0:
                    G[row[0]].add(row[1])
            else:
                G[row[0]] = set()
                if row[2] >= 3.0:
                    G[row[0]].add(row[1])
        dict_of_items = {} #dictionary of all movies rated by users
```

```
        for row in testset:
            if row[0] in dict_of_items.keys():
                dict_of_items[row[0]].append(row[1])
            else:
                dict_of_items[row[0]] = []
                dict_of_items[row[0]].append(row[1])
        mod_testset = [row for row in testset if (len(dict_of_items[row[0]]) >=␣
↪val and len(G[row[0]]) > 0)]
        res = SVD(n_factors=22,n_epochs=20,verbose=False).fit(trainset).
↪test(mod_testset)
        est_rat = {} #dictionary of estimated ratings by users
        for row in res:
            if row[0] in est_rat.keys():
                est_rat[row[0]].append((row[1],row[3]))
            else:
                est_rat[row[0]] = []
                est_rat[row[0]].append((row[1],row[3]))
        precision_u = []
        recall_u = []
        for item in est_rat.keys():
            S_all = est_rat[item]
            S_all = sorted(S_all,key=lambda x:x[1],reverse=True)
            S_t = set([row[0] for row in S_all[0:val]])
            precision_u.append(len(S_t.intersection(G[item]))/float(len(S_t)))
            recall_u.append(len(S_t.intersection(G[item]))/float(len(G[item])))
        precision_set.append(np.mean(precision_u))
        recall_set.append(np.mean(recall_u))
    prec_list_svd.append(np.mean(precision_set))
    rec_list_svd.append(np.mean(recall_set))
```

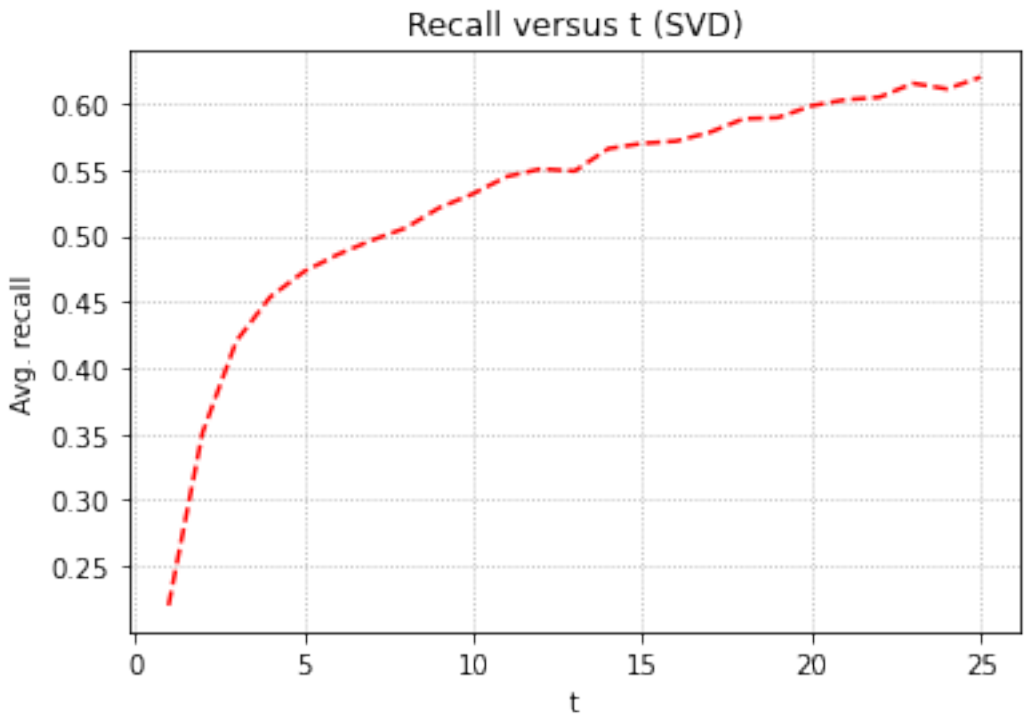
```
Testing for t = 1
Testing for t = 2
Testing for t = 3
Testing for t = 4
Testing for t = 5
Testing for t = 6
Testing for t = 7
Testing for t = 8
Testing for t = 9
Testing for t = 10
Testing for t = 11
Testing for t = 12
Testing for t = 13
Testing for t = 14
Testing for t = 15
Testing for t = 16
Testing for t = 17
```

```
Testing for t = 18
Testing for t = 19
Testing for t = 20
Testing for t = 21
Testing for t = 22
Testing for t = 23
Testing for t = 24
Testing for t = 25
```

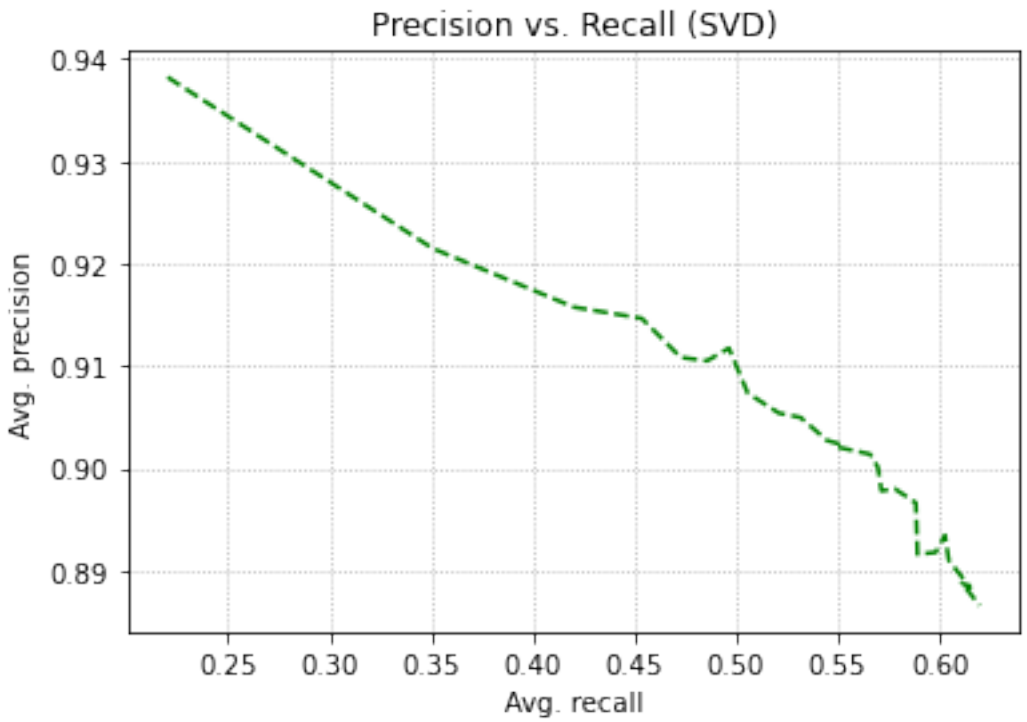
[317]:
```
plt.plot(t,prec_list_svd,linestyle='--',color='b')
plt.grid(linestyle=':')
plt.title('Precision versus t (SVD)')
plt.ylabel('Avg. precision')
plt.xlabel('t')
plt.savefig('Q38a.png',dpi=300,bbox_inches='tight')
plt.show()
```



[318]:
```
plt.plot(t,rec_list_svd,linestyle='--',color='r')
plt.grid(linestyle=':')
plt.title('Recall versus t (SVD)')
plt.ylabel('Avg. recall')
plt.xlabel('t')
plt.savefig('Q38b.png',dpi=300,bbox_inches='tight')
plt.show()
```

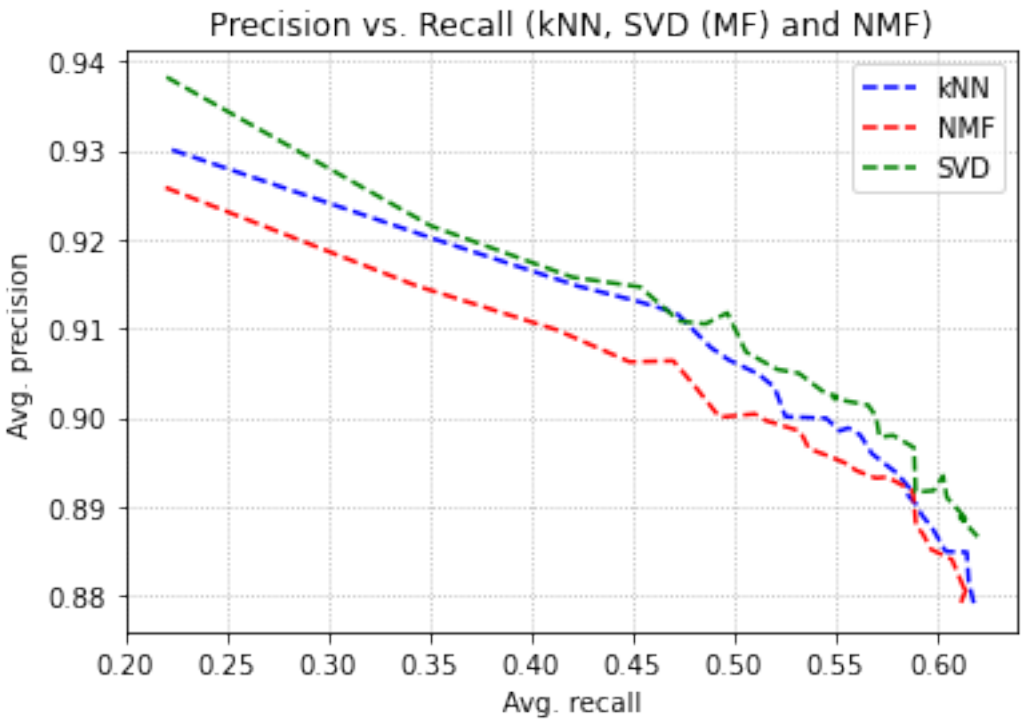


[319]:
```
plt.plot(rec_list_svd,prec_list_svd,linestyle='--',color='g')
plt.grid(linestyle=':')
plt.title('Precision vs. Recall (SVD)')
plt.ylabel('Avg. precision')
plt.xlabel('Avg. recall')
plt.savefig('Q38c.png',dpi=300,bbox_inches='tight')
plt.show()
```



29 Question 39

[320]:
```
fig, ax = plt.subplots()
ax.plot(rec_list_knn,prec_list_knn,linestyle='--',color='b',label='kNN')
ax.plot(rec_list_nmf,prec_list_nmf,linestyle='--',color='r',label='NMF')
ax.plot(rec_list_svd,prec_list_svd,linestyle='--',color='g',label='SVD')
plt.grid(linestyle=':')
plt.title('Precision vs. Recall (kNN, SVD (MF) and NMF)')
plt.ylabel('Avg. precision')
plt.xlabel('Avg. recall')
plt.legend(loc="best")
plt.savefig('Q39.png',dpi=300,bbox_inches='tight')
plt.show()
```


Precision vs. Recall (kNN, SVD (MF) and NMF)
kNN
NMF
SVD
Avg. precision
Avg. recall
0.94
0.93
0.92
0.91
0.90
0.89
0.88
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0.60

[ ]: