ECE 219: Data Mining - Models & Algorithms **Spring 2024**
Prof. Vwani Roychowdhury UCLA, Department of ECE

Project 3: Recommender Systems

Due May 17, 2024 by 11:59 pm

1 Note

In this project, you are provided only with the training subset of the movie dataset. All the metrics that you will report have to do with validation performance as averaged across folds. For the web10k dataset, you are allowed to directly use the partitioned dataset.

2 Introduction

The increasing importance of the web as a medium for electronic and business transactions and advertisement, and social media has served as a driving force behind the development of recommender systems technology. Among the benefits, recommender systems provide a means to prioritize data for each user from the infinite information available on the internet. Such systems are critical to ensuring (among others): (a) the detection of hate speech, (b) user retention on a web service, and (c) fast and high-quality access to relevant information. An important catalyst is the ease with which the web enables users to provide feedback about a small portion of the web that they traverse.

Such user-driven *sparse* feedback poses the following challenge in the desing of recommender systems: Can we utilize these sparse user datapoints to infer generalized user interests?

We define some terms:

- The entity to which the recommendation is provided is referred to as the *user*;
- The product being recommended is an *item*.

The basic models for recommender systems works with two kinds of data:

A User-Item interactions such as ratings (a user (you) provides ratings about a movie (item));

B Attribute information about the users and items such as textual profiles or relevant keywords (deep representations about a user or item).

Models that use type A data are referred to as collaborative filtering methods, whereas models that use type B data are referred to as content-based methods. In this project, we will build a recommendation system using collaborative filtering methods.

3 Collaborative filtering models

Collaborative filtering models use the collaborative power of established user-item interactions to make recommendations about new user-item interactions. In this project we use a ratings database where the user is an audience member who viewed a movie, and the item is the movie being rated.

The main challenge in designing collaborative filtering methods is that the underlying ratings matrices are sparse. Consider this example of a movie application in which users specify ratings indicating their like or dislike of specific movies. *Most users would have viewed only a small fraction of the large universe of available movies and as a result most of the ratings are unspecified.*

The basic idea of collaborative filtering methods is that these unspecified ratings can be imputed because the observed ratings are often highly correlated across various users and items.

For example, consider two users named John and Molly, who have very similar tastes. If their respective ratings exist within our database and are very similar, then the media recommended to them should likely be similar as well.

For those few scenarios in which only John has rated a movie M, the similarity across other movies to Molly's preferences should make clear that Molly might also prefer movie M. Thus, most collaborative filtering methods leverage either inter-item correlations or inter-user correlations for the prediction process.

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

1. **Neighborhood-based collaborative filtering**: Directly leverages the choices of other users to determine potential items to recommend to the current user.

2. **Model-based collaborative filtering**: Estimates a joint model from all user data, enabling the generation of new recommendations without accessing the entire user base, and allowing for queries on a more compact model.

4 Dataset

In this project, we will build a recommendation system to predict the ratings of movies in the provided dataset. The dataset can be downloaded here.

For the subsequent discussion, we assume that the ratings matrix is denoted by R (you will have to construct this), and it is an $m \times n$ matrix containing m users (rows) and n movies (columns). The (i, j) entry of the matrix is the rating by user i for movie j and is denoted by r_{ij}. Before moving on to the collaborative filter implementation, we will analyze and visualize some properties of this dataset.

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

A **Compute the sparsity of the movie rating dataset**:

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

B **Plot a histogram showing the frequency of the rating values**: Bin the raw rating values into intervals of width 0.5 and use the binned rating values as the horizontal axis. Count the number of entries in the ratings matrix R that fall within each bin and use this count as the height of the vertical axis for that particular bin. Comment on the shape of the histogram.

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

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

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

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

5 Neighborhood-based collaborative filtering

The basic idea in neighborhood-based methods is to use either user-user similarity or item-item similarity to make predictions from a ratings matrix. There are two basic principles used in neighborhood-based models:

1. *User-based models*: Similar users have similar ratings on the same item. Therefore, if John and Molly have rated movies in a similar way in the past, then one can use John's observed ratings on the movie *Terminator* to predict Molly's rating on this movie. *Item is kept constant.*
2. *Item-based models*: Similar items are rated in a similar way by the same user. Therefore, John's ratings on similar science fiction movies like *Alien* and *Predator* can be used to predict his rating on *Terminator*. *User is kept constant.*

In this project, we will only implement user-based collaborative filtering (implementation of item-based collaborative filtering is very similar).

5.1 User-based neighborhood models

In this approach, we are trying to find a set of users similar in their rating strategy to a target user. This results in a user-based neighborhood and we will use the majority vote within the neighborhood to provide recommendations.

In order to determine the neighborhood of the target user u, their similarity to all the other users is computed. Therefore, a similarity function needs to be created between each pair of the historical rating patterns - one by each user across the movies. In this project, we will use the Pearson-correlation coefficient to compute this similarity as a correlation.

5.2 Pearson-correlation coefficient

The Pearson-correlation coefficient between users u and v denoted by Pearson(u,v) captures the similarity between the rating vectors of users u and v. First some notation:

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

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

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

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

Then, with the above notation, the Pearson-correlation coefficient between a pair of users u and v is defined by equation 2:

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

5.3 k-Nearest neighborhood (k-NN)

Having introduced a similarity metric between users (as a correlation coefficient between their ratings across movies), we are now ready to define a neighborhood of users. The k-Nearest neighbors of user u, denoted by P_u, is the set of k users with the highest Pearson-correlation coefficient with user u (pairwise).

5.4 Prediction function

The predicted rating that user u might award for item j, denoted by $\hat{r}_{uj}$, can simply be modeled by equation 3:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u} Pearson(u,v)(r_{vj} - \mu_v)}{\sum_{v \in P_u} |Pearson(u,v)|} \tag{3}$$

QUESTION 3: Understanding the Prediction function: Can you explain the reason behind mean-centering the raw ratings ($r_{vj} - \mu_v$) in the prediction function? (Hint: Consider users who either rate all items highly or rate all items poorly and the impact of these users on the prediction function.)

5.5 k-NN collaborative filter

The previous sections have equipped you with the basics needed to implement a k-NN collaborative filter for predicting ratings of the movies. *Although we have provided you with the equations needed to write a function for predicting the ratings, we don't require you to write it. Instead, you can use the built-in python functions for prediction.*

5.5.1 Design and test via cross-validation

In this part of the project, you will design a k-NN collaborative filter and test its performance via 10-fold cross validation. In a 10-fold cross-validation, the dataset is partitioned into 10 equal sized subsets. Of the 10 subsets, a single subset is retained as the validation data for testing the filter, and the remaining 9 subsets are used to train the filter. The cross-validation process is then repeated 10 times, with each of the 10-subsets used exactly once as the validation data.

QUESTION 4: Design a k-NN collaborative filter to predict the ratings of the movies in the original dataset and evaluate its performance using 10-fold cross validation. Sweep k (number of neighbors) from 2 to 100 in step sizes of 2, and for each k compute the average RMSE and average MAE obtained by averaging the RMSE and MAE across all 10 folds. Plot average RMSE (Y-axis) against k (X-axis) and average MAE (Y-axis) against k (X-axis).

The functions that might be useful for solving this question are described in these documentations [1].

Use Pearson-correlation function as the similarity metric. You can read about how to specify the similarity metric in the documentation: `http://surprise.readthedocs.io/en/stable/similarities.html`

QUESTION 5: Use the plot from question 4, to find a 'minimum k'. Note: The term 'minimum k' in this context means that increasing k above the minimum value would not result in a significant decrease in average RMSE or average MAE. If you get the plot correct, then 'minimum k' would correspond to the k value for which average RMSE and average MAE converges to a steady-state value. Please report the steady state values of average RMSE and average MAE.

[1] `http://surprise.readthedocs.io/en/stable/knn_inspired.html`, `http://surprise.readthedocs.io/en/stable/model_selection.html#surprise.model_selection.validation.cross_validate`

5.6 Filter model performance based on subsets of the raw data

In this part of the project, we will analyze the performance of the k-NN collaborative filter in predicting the ratings of the movies in trimmed data subsets. The subsets can be formed in many ways, but we will consider the following trimming options:

- Popular movie trimming: In this trimming, we trim the dataset to contain movies that have received more than 2 ratings. If a movie in the set has received less than or equal to 2 ratings in the entire dataset then we delete that movie from the set and do not train/predict the rating of that movie using the model.
- Unpopular movie trimming: In this trimming, we trim the dataset to contain movies that have only received less than or equal to 2 ratings. If a movie in the set has received more than 2 ratings in the entire dataset then we delete that movie from the set and do not train/predict the rating of that movie using the model.
- High variance movie trimming: In this trimming, we trim the set to contain movies that have variance (of the rating values received) of at least 2 and have received at least 5 ratings in the entire dataset. If a movie has variance less than 2 or has received less than 5 ratings in the entire dataset then we delete that movie from the set and do not train/predict the rating of that movie using the model.

Having defined the types of trimming operations above, now we can evaluate the performance of the k-NN filter architecture in predicting the ratings of the movies in the trimmed dataset.

5.6.1 Performance evaluation using ROC curve

Receiver operating characteristic (ROC) curve is a commonly used graphical tool for visualizing the performance of a binary classifier. It plots the true positive rate (TPR) against the false positive rate (FPR).

In the context of recommendation systems, it is a measure of the relevance of the items recommended to the user. Since the observed ratings are in a continuous scale (0-5), so we first need to convert the observed ratings to a binary scale. This can be done by thresholding the observed ratings. If the observed rating is greater than the threshold value, then we set it to 1 (implies that the user liked the item). If the observed rating is less than the threshold value, then we set it to 0 (implies that the user disliked the item). After having performed this conversion, we can plot the ROC curve for the recommendation system in a manner analogous to that of a binary classifier.

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

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

- Sweep k (number of neighbors) from 2 to 100 in step sizes of 2, and for each k compute the average RMSE obtained by averaging the RMSE across all 10 folds. Plot average RMSE (Y-axis) against k (X-axis). Also, report the minimum average RMSE.
- Plot the ROC curves for the k-NN collaborative filters for threshold values $[2.5, 3, 3.5, 4]$. These thresholds are applied only on the ground truth labels in held-out validation set. For each of the plots, also report the area under the curve (AUC) value. You should have 4×4 plots in this section (4 trimming options – including no trimming times 4 thresholds) - all thresholds can be condensed into one plot per trimming option yielding only 4 plots.

We provide you with the following hints:

- Write trimming function that takes as input the set of data and outputs a trimmed set;

- For each value of k, split the trimmed dataset into 10 pairs of training and validation sets.

  (trainset 1, testset 1), (trainset 2, testset 2), . . . , (trainset 10, testset 10)

  The following documentation might be useful for the splitting:
  `http://surprise.readthedocs.io/en/stable/getting_started.html#use-cross-validation-iterators`
- For each pair of (trainset, testset):
  - Train the collaborative filter on the trimmed train set
  - Predict the ratings of the movies in the trimmed validation set using the trained collaborative filter
  - Compute the RMSE of the predictions in the trimmed validation set
- Compute the average RMSE by averaging across all the 10 folds.
- For the ROC plotting, split the dataset into 90% for training and 10% for validation. The functions described in the documentation below might be useful `http://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html`

6 Model-based collaborative filtering

In model-based collaborative filtering, models are developed using machine learning algorithms to predict users' rating of unrated items. Some examples of model-based methods include decision trees, rule-based models, bayesian methods, and latent factor models. In this project, we will explore latent factor based models for collaborative filtering.

6.1 Latent factor based collaborative filtering

Latent factor based models can be considered as a direct method for matrix completion. It estimates the missing entries of the rating matrix R, to predict what items a user will most probably like other than the ones they have rated. The basic idea is to exploit the fact that significant portions of the rows and columns of the rating matrix are correlated. As a result, the data has built-in redundancies and the rating matrix R can be approximated by a low-rank matrix. The low-rank matrix provides a robust estimation of the missing entries.

The method of approximating a matrix by a low-rank matrix is called matrix factorization. The matrix factorization problem in latent factor based model can be formulated as an optimization problem given by 4

$$\underset{U,V}{\text{minimize}} \quad \sum_{i=1}^{m}\sum_{j=1}^{n}(r_{ij} - (UV^T)_{ij})^2 \tag{4}$$

In the above optimization problem, U and V are matrices of dimension $m \times k$ and $n \times k$ respectively, where k is the number of latent factors. However, in the above setting it is assumed that all the entries of the rating matrix R is known, which is not the case with sparse rating matrices. Fortunately, latent factor model can still find the matrices U and V even when the rating matrix R is sparse. It does it by modifying the cost function to take only known rating values into account. This modification is achieved by defining a weight matrix W in the following manner:

$$W_{ij} = \begin{cases} 1, r_{ij} \text{ is known} \\ 0, r_{ij} \text{ is unknown} \end{cases}$$

Then, we can reformulate the optimization problem as

$$\underset{U,V}{\text{minimize}} \quad \sum_{i=1}^{m}\sum_{j=1}^{n}W_{ij}(r_{ij} - (UV^T)_{ij})^2 \tag{5}$$

Since the rating matrix R is sparse, the observed set of ratings is very small. As a result, it might cause over-fitting. A common approach to address this problem is to use regularization. The optimization problem with regularization is given by equation 6. The regularization parameter λ is always non-negative and it controls the weight of the regularization term.

$$\underset{U,V}{\text{minimize}} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} W_{ij}(r_{ij} - (UV^T)_{ij})^2 + \lambda \|U\|_F^2 + \lambda \|V\|_F^2 \tag{6}$$

There are many variations to the unconstrained matrix factorization formulation (equation 6) depending on the modification to the objective function and the constraint set. In this project, we will explore two such variations:

- Non-negative matrix factorization (NMF)
- Matrix factorization with bias (MF with bias)

6.2 Non-negative matrix factorization (NMF)

Non-negative matrix factorization may be used for ratings matrices that are non-negative. As we have seen in the lecture, the major advantage of this method is the high level of interpretability it provides in understanding the user-item interactions. The main difference from other forms of matrix factorization is that the latent factors U and V must be non-negative. Therefore, optimization formulation in non-negative matrix factorization is in 7:

$$\begin{aligned} &\underset{U,V}{\text{minimize}} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} W_{ij}(r_{ij} - (UV^T)_{ij})^2 + \lambda \|U\|_F^2 + \lambda \|V\|_F^2 \\ &\text{subject to} \quad U \geq 0, V \geq 0 \end{aligned} \tag{7}$$

There are many optimization algorithms like stochastic gradient descent (SGD), alternating least-squares (ALS),etc for solving the optimization problem in 7. Since you are familiar with the SGD method, we will not describe it here. Instead, we will provide the motivation and main idea behind the ALS algorithm. SGD is very sensitive to initialization and step size. ALS is less sensitive to initialization and step size, and therefore a more stable algorithm than SGD. ALS also has a faster convergence rate than SGD. The main idea in ALS, is to keep U fixed and then solve for V. In the next stage, keep V fixed and solve for U. In this algorithm, at each stage we are solving a least-squares problem.

Although ALS has a faster convergence rate and is more stable, we will use SGD in this project. The main reason behind this is based on the fact that the python package that we will be using to design the NMF-based collaborative filter only has the SGD implementation. This choice would have no effect on the performance of the filter designed because both the SGD and ALS converges for the original dataset. The only downside of using SGD is that it will take a little bit longer to converge, but that will not be a big issue as you will see while designing the NMF filter.

QUESTION 7: **Understanding the NMF cost function:** Is the optimization problem given by equation 5 convex? Consider the optimization problem given by equation 5. For U fixed, formulate it as a least-squares problem.

6.2.1 Prediction function

After we have solved the optimization problem in equation 7 for U and V, then we can use them for predicting the ratings.The predicted rating of user i for item j, denoted by $\hat{r}_{ij}$, is given by equation 8

$$\hat{r}_{ij} = \sum_{s=1}^{k} u_{is} \cdot v_{js} \tag{8}$$

Having covered the basics of matrix factorization, now we are ready to implement a NMF based collaborative filter to predict the ratings of the movies. We have provided you with the necessary background to implement the filter on your own, but we don't require you to do that. Instead, you can use provided functions in Python for the implementation.

6.2.2 Design and test via cross-validation

In this part, you will design a NMF-based collaborative filter and test its performance via 10-fold cross validation. Details on 10-fold cross validation have been provided in one of the earlier sections.

QUESTION 8: Designing the NMF Collaborative Filter:

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

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

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

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

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

For solving this question, the functions described in the documentation below might be useful: http://surprise.readthedocs.io/en/stable/matrix_factorization.html

6.2.3 Interpretability of NMF

The major advantage of NMF over other forms of matrix factorization is not necessarily one of accuracy, but that of the high level of interpretability it provides in understanding user-item interactions. In this part of the project, we will explore the interpretability of NMF. Specifically, we will explore the connection between latent factors and movie genres.

QUESTION 9: Interpreting the NMF model: Perform Non-negative matrix factorization on the ratings matrix R to obtain the factor matrices U and V, where U represents the user-latent factors interaction and V represents the movie-latent factors interaction (use $k = 20$). For each column of V, sort the movies in descending order and report the genres of the top 10 movies. Do the top 10 movies belong to a particular or a small collection of genre? Is there a connection between the latent factors and the movie genres?

In this question, there will be 20 columns of V and you don't need to report the top 10 movies and genres for all the 20 columns. You will get full credit, as long as you report for a couple columns and provide a clear explanation on the connection between movie genres and latent factors.

6.3 Matrix factorization with bias (MF with bias)

In MF with bias, we modify the cost function (equation 6) by adding bias term for each user and item. With this modification, the optimization formulation of MF with bias is given by equation 9

$$\underset{U,V,b_u,b_i}{\text{minimize}} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} W_{ij}(r_{ij} - \hat{r}_{ij})^2 + \lambda \|U\|_F^2 + \lambda \|V\|_F^2 + \lambda \sum_{u=1}^{m} b_u^2 + \lambda \sum_{i=1}^{n} b_i^2 \tag{9}$$

In the above formulation, b_u is the bias of user u and b_i is the bias of item i, and we jointly optimize over U, V, b_u, b_i to find the optimal values.

6.3.1 Prediction function

After we have solved the optimization problem in equation 9 for U, V, b_u, b_i, then we can use them for predicting the ratings. The predicted rating of user i for item j, denoted by $\hat{r}_{ij}$ is given by equation 10

$$\hat{r}_{ij} = \mu + b_i + b_j + \sum_{s=1}^{k} u_{is} \cdot v_{js} \tag{10}$$

where μ is the mean of all ratings, b_i is the bias of user i, and b_j is the bias of item j.

6.3.2 Design and test via cross-validation

In this part, you will design a MF with bias collaborative filter and test it's performance via 10-fold cross validation. Details on 10-fold cross validation have been provided in one of the earlier sections.

QUESTION 10: Designing the MF Collaborative Filter:

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

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

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

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

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

For solving this question, the functions described in the documentation below might be useful: `http://surprise.readthedocs.io/en/stable/matrix_factorization.html`

7 Naive collaborative filtering

In this part of the project, we will implement a naive collaborative filter to predict the ratings of the movies in the original dataset. This filter returns the mean rating of the user as it's predicted rating for an item.

7.1 Prediction function

The predicted rating of user i for item j, denoted by $\hat{r}_{ij}$ is given by equation 11

$$\hat{r}_{ij} = \mu_i \tag{11}$$

where μ_i is the mean rating of user i.

7.2 Design and test via cross-validation

Having defined the prediction function of the naive collaborative filter, we will design a naive collaborative filter and test its performance via 10-fold cross validation.

An important thing to note about the naive collaborative filter is that there is no notion of training. For training the model, split the dataset into 10 pairs of train set and validation set and for each pair predict the ratings of the movies in the validation set using the prediction function (no model fitting required). Then compute the RMSE for this fold and repeat the procedure for all the 10 folds. The average RMSE is computed by averaging the RMSE across all the 10 folds.

QUESTION 11: Designing a Naïve Collaborative Filter:

- Design a naive collaborative filter to predict the ratings of the movies in the original dataset and evaluate it's performance using 10-fold cross validation. Compute the average RMSE by averaging the RMSE across all 10 folds. Report the average RMSE.
- **Performance on dataset subsets:** For each of Popular, Unpopular and High-Variance test subsets -
  - Design a naive collaborative filter for each trimmed set and evaluate its performance using 10-fold cross validation.
  - Compute the average RMSE by averaging the RMSE across all 10 folds. Report the average RMSE.

8 Performance comparison

In this section, we will compare the performance of the various collaborative filters (designed in the earlier sections) in predicting the ratings of the movies in the original dataset.

QUESTION 12: Comparing the most performant models across architecture: Plot the best ROC curves (threshold = 3) for the k-NN, NMF, and MF with bias based collaborative filters in the same figure. Use the figure to compare the performance of the filters in predicting the ratings of the movies.

9 Ranking

In earlier sections, we explored collaborative filtering techniques for predicting ratings, effectively understanding user preferences and behavior from historical data, laying a foundational understanding of user behavior and item affinity. Now, we shift our focus to how ranking approaches enhance recommendation systems by not only identifying relevant items but also prioritizing them for optimal user satisfaction and engagement.

Next, we'll demonstrate the learning-to-rank concept using LightGBM [1] (Light Gradient Boosting Machine) with LambdaRank [2], a pairwise approach, applied to the Microsoft Learning to Rank (MSLR-WEB10K) dataset.

9.1 Problem Formulation for Learning-to-Rank

Learning to Rank (LTR) is a sophisticated area of machine learning that deals with the problem of ranking a set of items in the correct order based on their relevance to a particular query or user preference. It's a critical component in various applications like search engines, recommendation systems, and online advertising.

The primary objective of LTR is to construct a model that can predict the optimal ordering of items (such as documents, products, or search results) for any given query. This model is typically trained on a dataset where the relevances of items to different queries are known. Note that a query can be broadly defined to include user past history, session context and applied filters and so on.

9.2 Mainstream Learning-to-Rank Methods

Mainstream Learning-to-Rank approaches can be broadly categorized into three groups:

1. *Pointwise Approach:*
   - Treats ranking as a regression or classification problem.
   - Focuses on predicting a numerical or categorical score for individual items.
2. *Pairwise Approach:*
   - Views ranking as a problem of correctly ordering pairs of items.
   - Aims to minimize the number of incorrectly ordered pairs.

   Let's consider a practical example to understand this concept better:

   Assume for a given query, we have two items x_i and x_j with their true relevance scores labeled as y_i and y_j respectively. We employ a scoring function $f(x)$ that predicts relevance scores s_i and s_j for x_i and x_j, respectively. With the predicted scores, we now have a pair of items $(x_i,\, x_j)$ along with their scores $(s_i,\, s_j)$.

   To calculate the error based on the order of this pair of items, We define a loss function $\text{Loss}(s_i, s_j)$. We approximate this problem to a binary classification task.

   Here, we can define the true probability P_{ij} as follows:

$$P_{ij} = \begin{cases} 1 & \text{if } y_i > y_j, \\ 0 & \text{if } y_i < y_j. \end{cases} \tag{12}$$

   The predicted probability $\overline{P_{ij}}$ is calculated as:

$$\overline{P_{ij}} = \frac{e^{(s_i - s_j)}}{1 + e^{(s_i - s_j)}} \tag{13}$$

Loss(s_i, s_j) is defined as the cross-entropy loss between the P_{ij} and the $\overline{P_{ij}}$:

$$\text{Loss}(s_i, s_j) = -\left[P_{ij}\log(\overline{P_{ij}}) + (1 - P_{ij})\log(1 - \overline{P_{ij}})\right] \tag{14}$$

3. *Listwise Approach:*
   - Considers the entire list of items as the object of optimization.
   - Focuses on directly optimizing the order of the whole list

In this part, we will utilize the pairwise approach implemented in LightGBM. This method differs from the collaborative filtering techniques previously mentioned in two significant ways. First, it is a versatile framework capable of incorporating an extensive range of features, including new features that can be added through retraining. Secondly, the pairwise learning-to-rank approach adopts a more relative training objective, focusing on relative preference between two items rather than directly predicting a human-labeled score. This approach is more suitable for real-life scenarios where a strict ranking order from human annotator cannot be obtained, but a relative preference between items can still be derived.

9.3 Datasets for Learning-to-Rank Problems

Datasets used in learning-to-rank problems typically consist of queries, a list of documents associated with each query, and relevance judgments for each query-document pair. These relevance judgments are often graded on a scale, indicating how well each document satisfies the query. Additionally, each query-document pair is usually represented by a feature vector. This vector contains various features that describe the relationship between the query and the document, like TF-IDF scores, page ranks, user click-through rates, query length, etc.

MSLR-WEB10K Dataset

The MSLR-WEB10k Dataset, which is a benchmark dataset widely used in learning-to-rank research. It possesses the following key characteristics:

- Size and Composition: The dataset contains 10,000 queries. Each query is associated with a set of documents (web pages), and each of these documents has been judged for its relevance to the corresponding query.
- Relevance Judgments: The relevance of each document to a query is labeled on a scale from 0 (irrelevant) to 4 (perfectly relevant).
- Feature Vectors: Each query-document pair is represented by a 136-dimensional feature vector.

Below are two rows from MSLR-WEB30K dataset:

```
0 qid:1 1:3 2:0 3:2 4:2 ... 135:0 136:0
2 qid:1 1:3 2:3 3:0 4:0 ... 135:0 136:0
```

Each row corresponds to a query-url pair. The first column is relevance label of the pair, the second column is query id, and the following columns are features.

9.4 Evaluation Metric for Ranking: nDCG

nDCG is a popular metric used in information retrieval to measure the effectiveness of a ranking system. It evaluates the quality of the ranking by considering the position of relevant items in the search results. The core idea is that highly relevant items appearing lower in a ranking list should be penalized as the user is less likely to consider them. nDCG is calculated as follows:

- **DCG (Discounted Cumulative Gain):**

$$DCG@K = \sum_{i=1}^{K} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{15}$$

where rel_i is the relevance of the item at position i.

- **Normalized DCG (nDCG):**

$$nDCG@K = \frac{DCG@K}{IDCG@K} \tag{16}$$

Here, $IDCG@K$ is the ideal DCG for the best possible ranking.

Consider a search system that returns five documents for a specific query. The relevance of each document to the query is graded on a scale from 0 (irrelevant) to 3 (highly relevant). For instance, assume the relevance scores for the five documents in the order returned by the system are: 3, 2, 3, 0, 1. The ideal order, in this case, would be: 3, 3, 2, 1, 0.

1. **Calculate DCG@5:**

$$\begin{aligned} DCG@5 &= \frac{2^3-1}{\log_2(1+1)} + \frac{2^2-1}{\log_2(2+1)} + \frac{2^3-1}{\log_2(3+1)} + \frac{2^0-1}{\log_2(4+1)} + \frac{2^1-1}{\log_2(5+1)} \\ &= 12.78 \end{aligned} \tag{17}$$

2. **Calculate IDCG@5:**

$$\begin{aligned} IDCG@5 &= \frac{2^3-1}{\log_2(1+1)} + \frac{2^3-1}{\log_2(2+1)} + \frac{2^2-1}{\log_2(3+1)} + \frac{2^1-1}{\log_2(4+1)} + \frac{2^0-1}{\log_2(5+1)} \\ &= 13.35 \end{aligned} \tag{18}$$

3. **Calculate nDCG@5:**

$$\begin{aligned} nDCG@5 &= \frac{DCG@5}{IDCG@5} \\ &= 0.957 \end{aligned} \tag{19}$$

QUESTION 13: Data Understanding and Preprocessing:

- Use the provided helper code for loading and pre-processing Web10k data.
- Print out the number of unique queries in total and show distribution of relevance labels.

QUESTION 14: LightGBM Model Training:

For each of the five provided folds, train a LightGBM model using the 'lambdarank' objective. After training, evaluate and report the model's performance on the test set using nDCG@3, nDCG@5 and nDCG@10.

QUESTION 15: Result Analysis and Interpretation:

For each of the five provided folds, list top 5 most important features of the model based on the importance score. Please use model.booster_.feature_importance(importance_type='gain') as demonstrated here for retrieving importance score per feature. You can also find helper code in the provided notebook.

QUESTION 16: Experiments with Subset of Features:

For each of the five provided folds:

- Remove the top 20 most important features according to the computed importance score in the question 15. Then train a new LightGBM model on the resulted 116 dimensional query-url data. Evaluate the performance of this new model on the test set using nDCG. Does the outcome align with your expectations? If not, please share your hypothesis regarding the potential reasons for this discrepancy.
- Remove the 60 least important features according to the computed importance score in the question 15. Then train a new LightGBM model on the resulted 76 dimensional query-url data. Evaluate the performance of this new model on the test set using nDCG. Does the outcome align with your expectations? If not, please share your hypothesis regarding the potential reasons for this discrepancy.

Submission

Please submit your **report**, and your **codes** with a **readme file** on how to run your code to Gradescope/BruinLearn. Only one submission per team is required. If you have any questions you can contact the TAs or post on Piazza.

References

[1] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Vol. 30. Curran Associates, Inc., 2017. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf`

[2] `https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/`