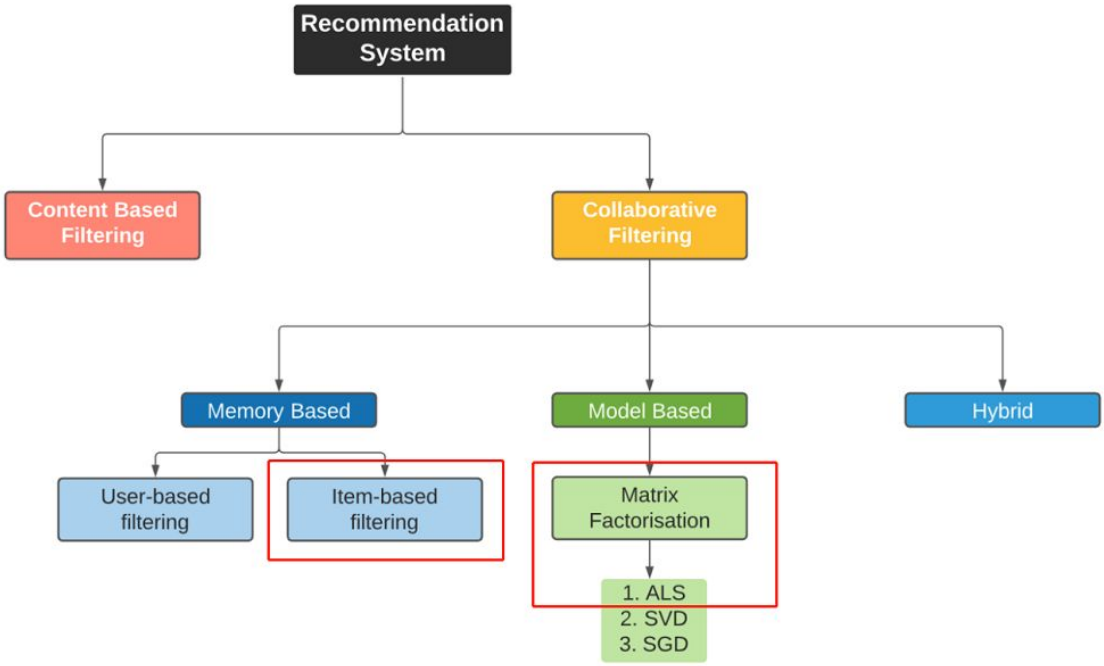
Analytical Report of Movie Recommendation

Group DLN: Chenxi Di, Tianqi Lou, Nan Tang

Objective

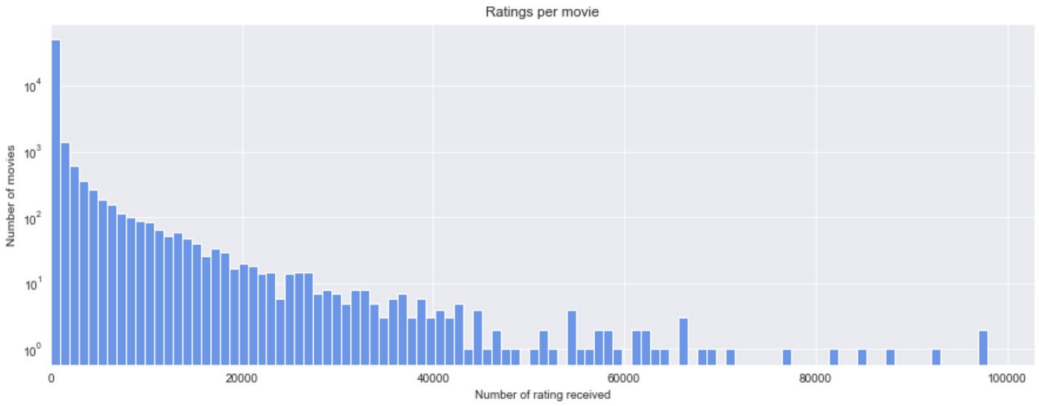
Recommender systems are information filtering tools that aspire to predict the rating for users and items, predominantly from big data to recommend their likes. Movie recommendation systems provide users with a mechanism assistance in classifying them with similar interests. This makes recommender systems essentially a central part of websites and e-commerce applications. Our practice focuses on collaborative filtering whose primary objective is to suggest a recommender system through KNN and ALS matrix factorization.



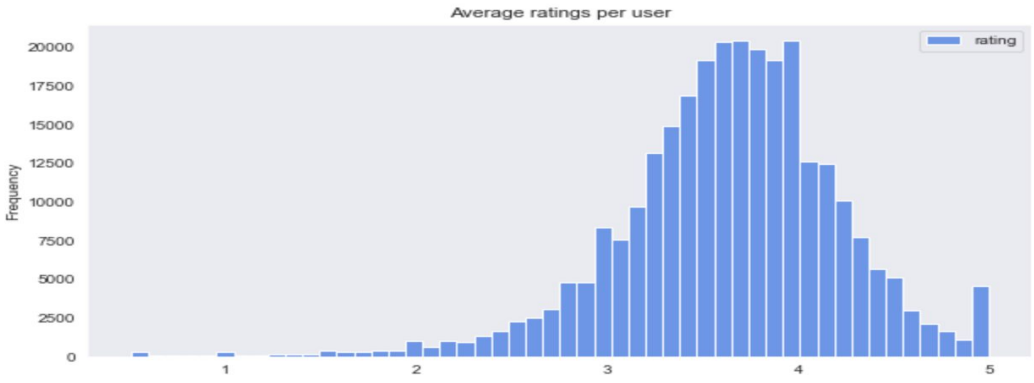
In the first half of this research report, we introduced an improved KNN model that speeds up prediction time than conventional implements. We have also implemented a model evaluation method for KNN models named Hit Rate, which we will discuss in the following sections. The second half of this report gives an account of our systematic approaches to optimize ALS model via cross validation, implementation of precision and recall evaluation, and analyze performance of ALS model on different sizes of the training data. It is hard for a collaborative filtering system to deal with cold start problems, therefore, our objective user groups are those who have rated at least one movie.

Data and Methods

We load rating data from the MovieLens, each row consisting of a user, a movie, a rating and a timestamp. (https://grouplens.org/datasets/movielens/latest/) Our group has built two collaborative filtering algorithms to recommend movies to users based on historical data. Below figure shows the distribution of ratings per movie. Most of the movies in the data received less than 10000 ratings.



Below shows the distribution of average rating scores per user. Most users give ratings between 3 and 5.



Our strategy in collaborative filtering is to filter the movies with more than 100 ratings. So the system is biased towards movies with more user interactions. As a new movie is added to the list, it will therefore rarely occur as a recommendation. Besides including accuracy scores such as hit rate for KNN model, root mean squared error, precision and recall for Matrix Factorization model, we also include evaluation metrics such as coverage (percent of possible recommendations that system can predict). Both of the models cannot give recommendations to users that have not rated any movies since we do not have user information. Constrained by runtime, we did not build very complex models. Current evaluation scores have indicated both models performed fairly well. However, in order to further evaluate the model performance, we suggest incorporating more implicit feedback or users' reaction to our recommendations, including browsing history, search patterns, or even mouse movements.

Item Memory-based Collaborative Filtering (KNN)

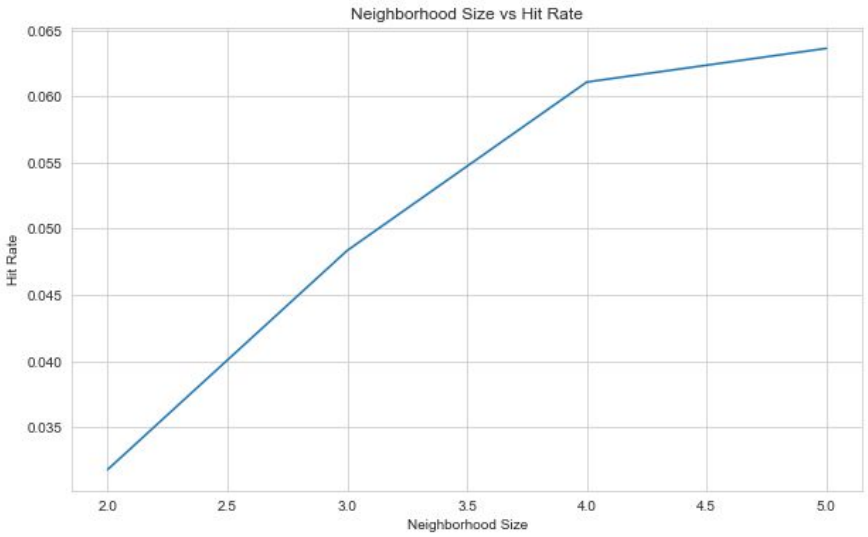
Model Implement: For memory based CF, we implemented k-nearest neighbors (KNN) algorithm to calculate the “distance” between the target movie and every other movie. We return the top k nearest neighbor movies as the most similar movie recommendations. In this Item-Item Collaborative Filtering, we recommend top 20 movies using criteria that “Users who liked this item also liked …”. For example, through our algorithm, for a certain user(id 281940), we give recommendations as below:

```
recommend movies: [Seven (a.k.a. Se7en) (1995), Remains of the Day, The
(1993), Reservoir Dogs (1992), Raiders of the Lost Ark (Indiana Jones and
the…), Full Metal Jacket (1987), Jaws (1975), Grosse Pointe Blank (1997),
Fifth Element, The (1997), My Cousin Vinny (1992), Lock, Stock & Two
Smoking Barrels (1998), Snatch (2000), Amelie (Fabuleux destin d'Amélie
Poulain, Le), Minority Report (2002),  Kill Bill: Vol. 1 (2003), Lord of
the Rings: The Return of the King, The...]
```

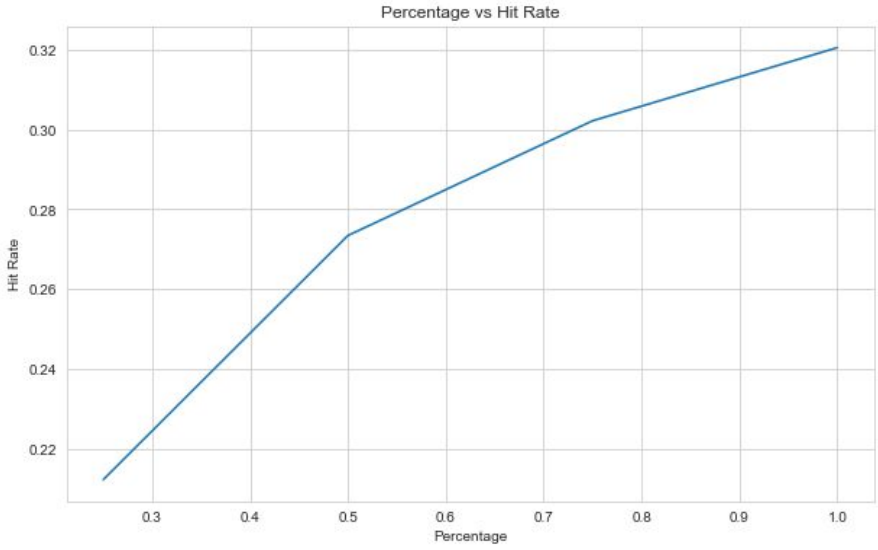
In the way of selecting neighbor movies, we start from the list of high-scored movies that are rated by given users. We use 2 as the rating threshold in the default setting, indicating that if a certain movie’s rating standardized by user preference is above 2 it would be labeled as ‘relevant/favorite’.

Accuracy: We used Hit Rate as an offline evaluation metric in this memory based model, which is defined as HIT RATE = (HITS IN TEST) / (NUMBER OF USERS). We implement the leave-one-out method to calculate the hit rate. First we find the movies a user has rates, and randomly take one movie out. Then we feed the rest of the movies into the pre-trained kNN model to see if the left-out movie is in the top-N recommendation. It is a “HIT” if it is in the top-N recommendation.

Different Size of Neighborhood: We test our model on different neighborhood sizes using hit rate metric on the smaller testing dataset. Since the dataset is small, the hit rate is very low, but we can still see the increasing trend of hit rate as neighborhood size increases. So we decide to use n_neighbors = 5 for the rest of the model. By this, we can get a relatively accurate model and avoid overfitting and redundancy.



Different Size of Data: We also test our model on different data sizes using hit rate metric. From the plot below, we can tell that as the data size increases, the hit rate increases, which also means that the accuracy of our model increases. It is also noticeable that our hit rate is not extremely high, and we think it is a good indication that there is certain space in our model for novelties and serendipities.



Coverage: The coverage of our model on the full data set is 857 out of 1000 movies.

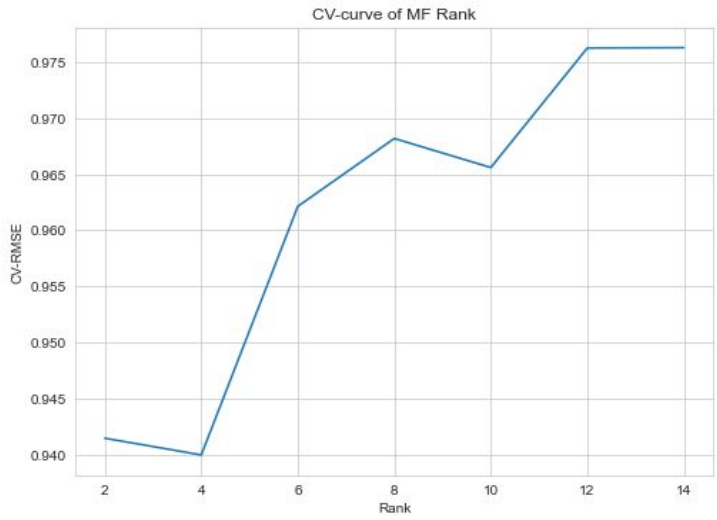
Model-based Collaborative Filtering (ALS)

For model based CF, users and products are described by a small set of latent factors, and we implemented Alternating Least Squares (ALS) algorithm to learn these latent factors.

Model Implement: In the python script, we implement the ALS model in the function "MF-recommendation". Input of this function includes a pre-trained matrix factorization model, a spark dataframe that contains rating information, and number of recommendations for each user which default as 20. Outputs of this recommender are stored in a python dictionary with user ID as key and a list of movie ID as value. Here we listed recommendations for the first three users.

```
userId: 231287
recommend movies: [5224, 7153, 4973, 771, 86504, 5772, 800, 72104, 50742,
2324, 1357, 515, 171011, 3920, 58425, 1293, 103867, 1295, 1358, 72226]
userId: 243392
recommend movies: [171011, 159819, 1198, 167832, 47, 121374, 119153,
48516, 112556, 147330, 65133, 73681, 55721, 96488, 4011, 6808, 116411,
86504, 2324, 89492]
userId: 37482
recommend movies: [171011, 159819, 5224, 4973, 86504, 50742, 147330,
58425, 2324, 1212, 720, 750, 3134, 800, 951, 26903, 177765, 1293, 1284,
7153]
```

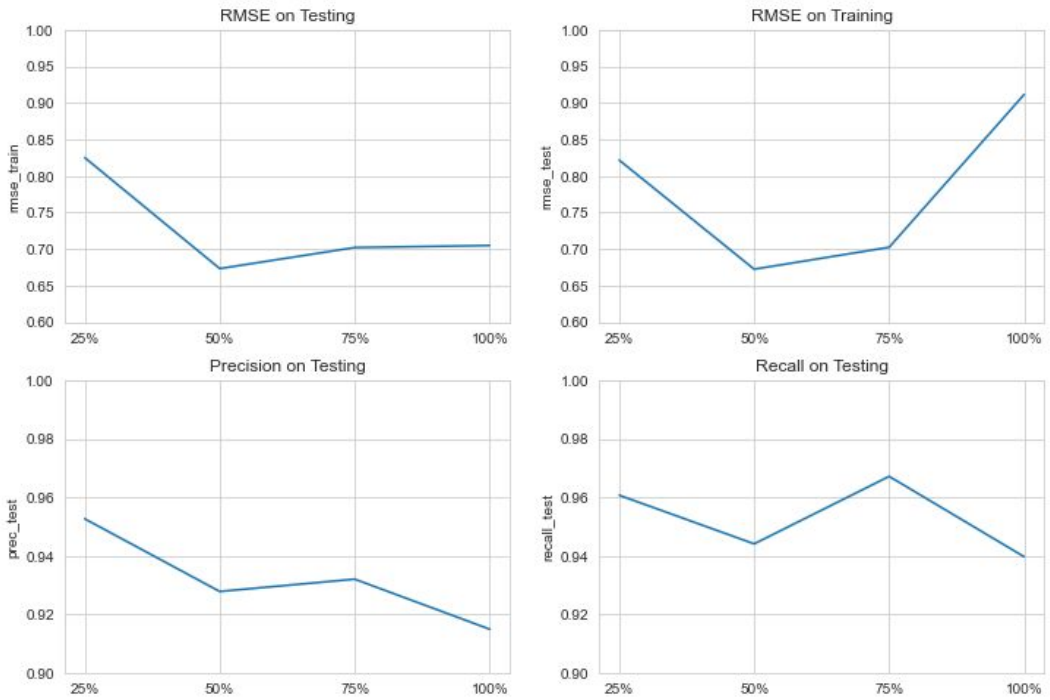
Optimization: Hyperparameters for ALS model include rank of latent matrix, maximum number of iteration, and regularization parameter. Considering the expensive time cost to perform k-fold cross-validation on data set with one thousand movies and two million ratings, we set maximum number of iteration and regularization penalty as default values and primarily focused on searching for best value of matrix rank. Brute force searching from rank value of two to fifteen shows when dimension is 4, the ALS model achieves its lowest RMSE on validation set.



Accuracy: Precision and recall are secondary evaluation metrics. We define predicted rating as positive for predicted scores greater than 2.5, i.e. true positive cases are when both user rating and predicted rating higher than 2.5. Similarly, we define false positive when user rating is lower than 2.5 but prediction is higher. As a result, our model achieved great scores on both precision and recall: they are 0.915 and 0.94 respectively. Precision implies users like 91.5% of movies that are predicted to have a high rating. Recall implies 94% of movies of which users are fond have a high predicted rating.

Coverage: Item coverage of the ALS model on the entire dataset is 30.8%. This fact indicates recommendations for all groups of users only include 308 movies out of 1000. Compared to item coverage of the KNN model, the ALS model risks more from popularity bias. User coverage of the ALS model is higher than 99.9%, indicating personalization serves for almost all users who have historical data reflected on rating.

Different Size of Data: As we change the data size from 25% to 50%, 75%, 100%, root mean squared error in the test dataset changes from 0.83 to 0.7, indicating that the model performs slightly better. But notice that RMSE in the training set increases to 0.9, starting to exceed the test RMSE. This implies the possibility of underfitting of our model. Due to the expensive cost of run time and space, we did not cross validate and tune other hyperparameters such as ALS max iterations(default as 10), regularization parameters(default as 0.1). As shown in the precision and recall graphs, the model experiences similar reduced accuracy scores as data becomes larger. Precision drops from 0.96 to 0.92 despite, and recall drops from 0.96 to 0.94 although there are some spikes. As the data scales, the run time takes extremely long especially when we attempt to transform pyspark dataframe to pandas dataframe.



Discussion

We have achieved the majority of our objectives, providing accurate and precise movie recommendations for existing users. In the improved KNN model, the leave one out hit rate is persuasive enough that the model captures the active user's taste of the movie. Precision and recall of the ALS model are both above 90%. As an aspect of accuracy, we are confident to deploy our recommender system to real applications.

Improvement could be done on scalability and run time. As the count of users and movies increases, the rating matrix we need to handle in the memory based approach has many values being 0. Such a sparse matrix costs lots of space and run time. As discussed in the model based approach, the hyperparameters we employed probably have not captured enough features in the data, which further results in reduced accuracy. But overall, based on hit rate and precision scores, both of the models perform fairly well. In the future, we hope to optimize the model further to achieve better performance and maybe incorporate other data sets to study temporal effects.

Reference

PySpark Collaborative Filtering with ALS, Snehal Nair, August 10
https://towardsdatascience.com/build-recommendation-system-with-pyspark-using-alternating-least-squares-als-matrix-factorisation-ebe1ad2e7679