Book Recommender System Kaggle Competition

Name: Kevin Chuang, **Kaggle ID:** kevinchuang8

INTRODUCTION

Recommender systems seek to predict a rating or preference a user will give certain items, such as a movies, books, songs, and products in general. The objective of this Kaggle competition assignment is to develop a recommender system for a medium-sized dataset that will accurately predict the rating that a user will give to a book given past their past ratings, and ultimately provide recommendations of books to a user based on their predicted ratings. The accuracy of the predicted ratings is determined by the root mean squared error (RMSE) to determine how close or far off a predicted rating is from the actual rating.

KAGGLE RANK & RMSE

At the time of writing this report, my current rank on the Kaggle competition public leaderboard is **1st** with a RMSE of **1.50613** on 30% of the test set. I achieve around a **5.5%** improvement from the baseline result of **1.59425**. The final leaderboard will be based on the other 70% of the test set.

EXPLORATORY DATA ANALYSIS

The training set contains 700,000 instances with three columns (user ID, book ID, and rating). The test set contains 300,000 instances with two columns (user ID, book ID). Additional book metadata is provided with information about each books' author, average rating, description, number of pages, etc. A majority of the exploratory data analysis is done on the training set with some analysis done on the additional book metadata. For the training set, there are 35,280 unique users and 68,371 unique books. The ratings lie within the range of 0 to 5 inclusive with a majority of the ratings being 0, 4 or 5 and the minority of the ratings being 1 or 2. The minimum number of ratings per user and the minimum number of ratings per book were both 1. Also, there were 25,851 books out of the 68,371 books that were only rated by 1 user. The majority of the number of ratings per book was very low and was in the 1 to 5 range. The majority of the number of ratings per user was also very low and was in the 1 to 16 range. The maximum number of ratings per book is 11,204, and the maximum number of ratings per person is 4,287. Thus, this dataset is severely imbalanced. An important observation of the training set was that there was duplicated data for some of the user and book pairs. For more details, see the *eda.ipynb* jupyter notebook.

DATA PREPROCESSING

From the exploratory data analysis, I found there were around 969 pairs of duplicated user and book pairs, which can create noise in our modeling. Thus, I simply removed the duplicates by taking the first instance, since the rating values were the same. I split 20% of the data in the training set to use as a hold-out test set, and use the remaining 80% of the training data for training and tuning the hyperparameters using k-fold cross validation. Since I mainly use matrix factorization collaborative filtering with latent factor models, there is minimal data preprocessing done. However, I did try the neighborhood-based collaborative filtering approaches (user-based and item-based). Based on the nearest-neighbor algorithm, the ratings were either mean-centered, standardized, or baseline-centered using the item or user mean and standard deviation. For example, for the user-based k-NN algorithm with mean-centering, the mean ratings for a user are subtracted from the user's original rating, and this adjusted rating is utilized in the prediction phase.

ALGORITHMS & HYPERPARAMETER TUNING

Collaborative filtering uses past transactions of users and ratings to establish connections between users and items. Two successful approaches include neighborhood based models that analyze similarities between users or items and latent factor models that directly profile users and items using latent factors (e.g. hidden variables). To achieve competitive results, I benchmarked different collaborative filtering approaches including matrix factorization algorithms and nearest neighbor algorithms. For the nearest neighbor algorithms, I experimented with k-nearest neighbor algorithms that use mean-centering, z-score normalization, and baseline ratings. For the matrix factorization algorithms, I experimented with singular value decomposition (SVD), singular value decomposition with implicit ratings (SVD++), and non-negative matrix factorization (NMF).

For all experiments, the hyperparameters of each algorithm (i.e. nearest neighbor based or matrix factorization based) were tuned using 5-fold cross validation. However, the optimal hyperparameters were searched differently based on the type of algorithm. Since the k-nearest neighbor algorithms contained a relatively more discrete set of parameters and were extremely memory intensive, I used a simple grid search over the maximum number of nearest neighbors, the distance metric, the minimum support of users or items, and the type of similarities computed (e.g. item-based or user-based). The parameters for the matrix factorization algorithms were continuous float values, such as the regularization and the learning rate. Because of this, I decided to use Bayesian optimization to search for the optimal parameters of each algorithm. Essentially, Bayesian optimization fits a surrogate model (e.g. a probabilistic Gaussian process) to the hyperparameters of an algorithm in order to search for the most optimal hyperparameters that minimize an objective function (e.g. RMSE). I essentially specify the bounds of each parameter of the algorithm and run the optimization operation for 200 iterations. For more details, see the *gp_minimize_svd.py* script.

For the final model, I use a weighted ensemble hybrid recommender system combining the best performing k-nearest neighbor algorithm using baseline ratings with the best performing SVD algorithm. The best performing algorithms and parameters were determined in terms of 5-fold cross validation RMSE. Essentially, the predictions are weighted and combined using an aggregation function, such as weighted average. The weights were also tuned using 5-fold cross validation and grid search to find the best performing ensemble hybrid recommender system. The tuning jobs and final submission jobs are run as batch jobs on HPC.

FINAL EXPERIMENTAL RESULTS

To ensure consistency, all experiments are run with the same random seed (e.g. 8). The results of each experiment are shown in the table below. The best hyperparameters are chosen for each of the algorithms based on the 5-fold cross-validation RMSE results. The model is then evaluated on the hold out test set (20% of the original data). The last column shows the Kaggle submission RMSE results from 30% of the test data. The final hybrid system contains the SVD algorithm with a weight of 0.7 and the KNN Baseline algorithm with a weight of 0.3 and achieves the lowest RMSE score for all columns. The optimal weights for the SVD algorithm and the optimal weights for the KNN Baseline are located in the *optimal_weights.json*.

Model	5-Fold CV RMSE	Test RMSE	Kaggle Submission
SVD	1.5589	1.5413	1.52183
SVD++	1.6272	1.6187	-
NMF	1.5925	1.5851	-
KNNBaseline	1.6153	1.5829	-
KNNWithMeans	1.6841	1.6560	-
KNNWithZScore	1.6832	1.6580	-
SVD + KNNBaseline	**1.5462**	**1.5268**	**1.50613**

SOURCES & LIBRARIES

For this Kaggle competition, I used Python and the surprise library for its implementations of different recommender system algorithms. In addition, I used some of the surprise Python tutorials in my code, such as splitting the train and test set. I used the skopt library for its implementations of Bayesian optimization and some of the code from its tutorial.