RECOMMENDER SYSTEMS

ANIME RECOMMENDATION

Yuehchao Wu

Table of Contents

Description of the Data

In this section, I will give an overview of the data chosen for my evaluation experiment including brief information of the original data, issues of the data, transformations applied to resolve the issues, and description of the final form of the data used for the project.

Brief Information of the Original Data

The source of the data is from Kaggle.com (Anime dataset from Kaggle). There are two associated datasets, rating dataset and anime dataset. The rating dataset contains 7,813,737 Ratings (Rating scale: 1-10) from 73,516 users on 12,294 anime with a density of 0.92%; the anime dataset consists of information about each anime with 7 columns (anime_id, name, genre, type, episodes, rating, and members).

The rating data is the main dataset for the experiment and the anime data is utilized as extra sources for data exploratory and preprocessing.

Issues of Data & Data Transformations

From the Exploratory Data Analysis, I discovered some issues of the data that made it hard to work with SURPRISE and could potentially disturb the credibility of the evaluation result. Therefore, the data transformations were applied to resolve the issues before the evaluation process started.

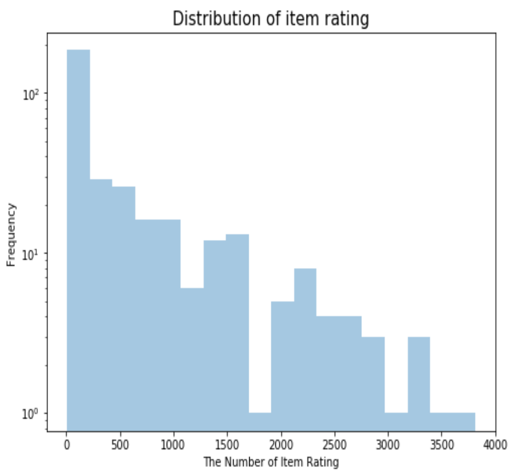
To elaborate, first of all, in order to reduce the size of data to a workable form for SURPRISE, I subsetted the data to focus only on movie-comedy based anime and treated the observations that have rating equal to -1 (Watch = True but no explicit rating) as unknown rating because trying to impute a value would just introduce errors. Secondly, in order to limit the experiment scope further and remove observations that do not contribute useful information, I wrote a function that utilized While Loop to iteratively eliminate the users or items that have less than 10 ratings and users that only use one rating scale on everything until there are no such observations in the data.

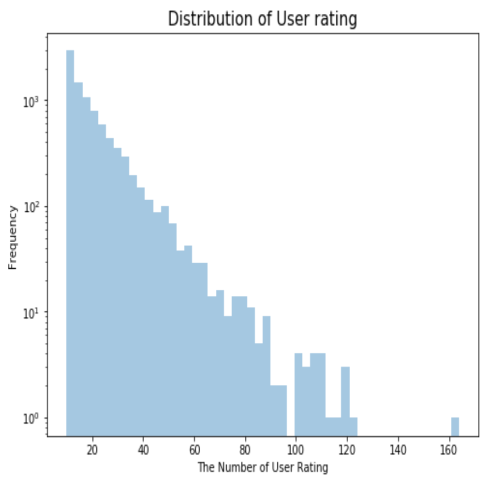
<u>Descriptive Statistic of The Final Form of The Rating Data</u>

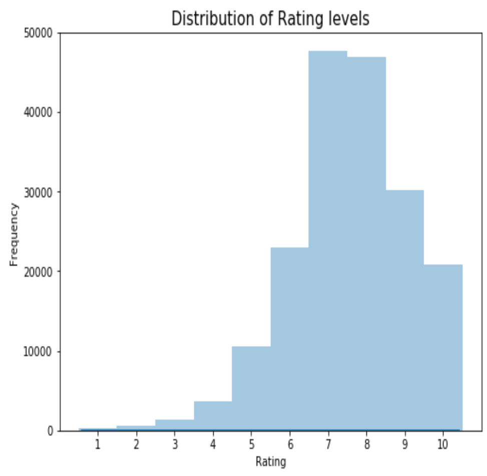
After transformation, the final form of the rating data used for the project contains 184,949 Ratings (Rating scale: 1-10) from 8,905 users on 334 anime with a density of 6.22%. The properties of the original rating data and the final form of the rating data are displayed in the following table.

	Number of Rating	**Number of user**	**Number of Item**	**Missing value (Rating = -1)**	**Number of Item less than 10 rating**	**Number of User less than 10 rating**	**Number of user that only rate with one value**	**Density**
Original Anime Rating data	7813737	73515	11200	1476496	2562	14482	5476	0.0092
Final Form of Anime Rating data	184949	8905	334	0	0	0	0	0.0622

It clearly shows that, comparing with the original rating data, the final form of the rating data is less sparse (density increase from 0.02% to 6.22%) and easier to work with (appropriate sized). Most importantly, evaluation on the final form of the rating data would be more accurate (no item or user has less than 10 ratings, and no extremely bias user) than on the original rating data.







Approach

This section will outline the defined recommendation task and evaluation goal and will also discuss the evaluation approach including methodology, metric, and algorithms of interest.

Defined Recommendation Task & Evaluation Goal

Before evaluating how well each recommendation system performs, it is important to define an appropriate recommendation task and set an evaluation goal accordingly. For me, finding a short ranked list of unique and relevant anime for each individual user would be a more meaningful recommendation task than simply predicting a future rating correctly. The evaluation goal, then, will focus on evaluating each experimented algorithm's ability to accomplish the defined task.

Evaluation Approach

Methodology

The Offline Evaluation was conducted as the evaluation methodology for the experiment due to the fact that the data were pre-collected and I had no access to the real users. The idea is to withhold part of the data as testing data for evaluation and use the rest as training data for model generalization. For the experiment, in order to get the most accurate evaluation estimation, I withheld 20% of the data and 20 % of unknown rating as the major testing set for evaluating recommendation accuracy (main goal). The rest of the known rating data was utilized to build tuned model that minimize unbiased RMSE using Five Fold Cross Validation and Grid Search. Noted, the stratification was applied during the Train-Test Split to ensure all users had a profile on both training and testing data. Also, testing set without unknown rating is utilized to evaluate RMSE as an aid for the evaluation. The properties of the training set and the testing sets are shown in the following table.

	Number of ratings	Number of Users	Number of Items
Training data (80% of data)	147,975	8,905	334
Testing data (20% of data)	36,990	8,905	329
Main testing data (20% of data and 20% unknown rating)	594,855	8,905	334

Evaluation Metrics

There are only predictive accuracy metrics, such as RMSE, available for evaluation in SURPRISE. However, it is not sufficient when evaluation goal focus on how good a system does at recommending short ranked lists of relevant anime for users. Therefore, the Rank Accuracy Metric (nDCG), and other metric (Average List Popularity and Coverage) are implemented (see Evaluation_Implemntation.py for detail) following the SURPRISE accuracy coding style (see accuracy.py on surprise github page as reference).

Briefly, a set of predictions is calculated on the test set with unknown rating included using SURPRISE test method. Then, the set of predictions is passed into modified version of 'get_top_n' function with a defined predicted rating threshold to get a dictionary of Top N Top N Recommendation. Then, each implemented metric is computed based on the information from the dictionary of Top N Recommendation ('uid' as key, a List of Top N Tuple ('iid','rui','est') as value for each 'uid'). Brief descriptions of the implemented metric are listed as following:

- **nDCG:** for evaluating the rank accuracy of a Top N Recommendation; scale 0 to1, the higher the value is the more accurate the rank.
- **Coverage:** for evaluating the percentage of items or users a system is able to cover; scale 0 to 1, the higher the value is, the better coverage.

  Math: # of user or item a system cover / number of user or items
- **Average List Popularity:** for evaluating how well a model does at recommending unpopular items; with no normalized scale, the lower the value is the better a system is at recommending long tail items.

  Math: average # of ratings for all items on the list.

The Algorithms of Interest

The Collaborative Filtering (CF) algorithms, item-based KNNWithMeans, SVD, Co-clustering, and SVDpp, were selected as my algorithms of interest and then each algorithm was tuned to its optimum using Grid Search. The CF algorithms were preferred over others in my study for couple of reasons. First of all, algorithms relied on properties of items, such as content-based, would simply not work well for the data since we did not have much information on anime attributes (already subset the rating data on comedy-movie anime only during size reduction). Secondly, it was simply more efficient since there were already some ready-to-use CF algorithms in the python library SURPRISE.

The chosen algorithms and its corresponding investigated hyperparameter values are listed as following:

- **Item-based KNNWithMeans:**

K	**Min_k**	**Name**	**Use_based**	**Min_support**	**Shrinkage**
[5,19,40,95,175]	1	[cosine, person]	False	[10,100,200]	100

- **SVD:**

n_factors	**n_epochs**	**biased**	**init_mean**	**init_std_dev**	**lr_all**	**reg_all**
[1,5,10,100,300]	[5,20]	True	0	0.1	[0.002, 0.02,0.2]	[0.002, 0.02, 0.2, 0.8]

- **Co-clustering:**

n_cltr_u	**n_cltr_i**	**n_epochs**
[5,10,100,300,3000]	[5,15,20,150]	[5, 20]

- **SVDpp:**

n_factors	**n_epochs**	**init_mean**	**init_std_dev**	**lr_all**	**reg_all**
[1,5,10,100,300]	[5,20]	0	0.1	[0.002, 0.02,0.2]	[0.002, 0.02, 0.2, 0.8]

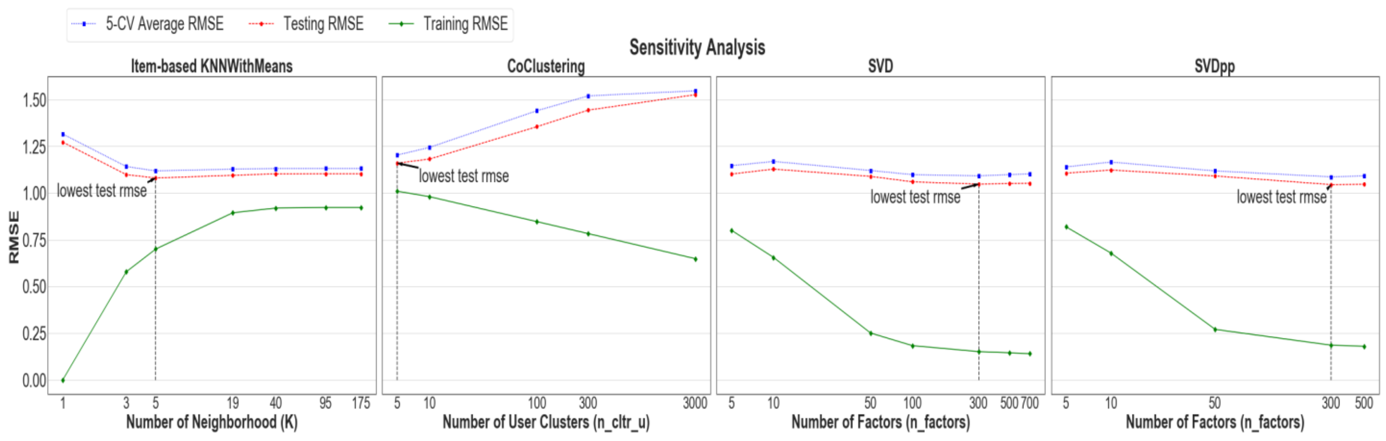
Briefly, my strategy of grid search is based on couple of rules. First of all, I tried to keep the searching space wide for each parameter in initial grid-search, then, if necessary, another grid search would be conducted with narrower range around the optimal hyperparameter found from first round. Secondly, the default value for each parameter is included and for the parameter that related to number of user or anime (such as K, n_factors…and so on), the square root of total number of user or item is also included. Lastly, these values are used as median and then the values around them are chose to investigated following the wide space rule. This strategy allowed me to investigate wider range of parameter combination and then narrow down the choice later.

Results

In this section, I will provide an example of Hyperparameter Sensitivity Analysis for each algorithm and describe the result by presenting evaluation values obtained from testing each tuned algorithm's recommendation performance on the testing set with unknown rating included. The major challenge of completing the project will also be described briefly.

Sensitivity Analysis

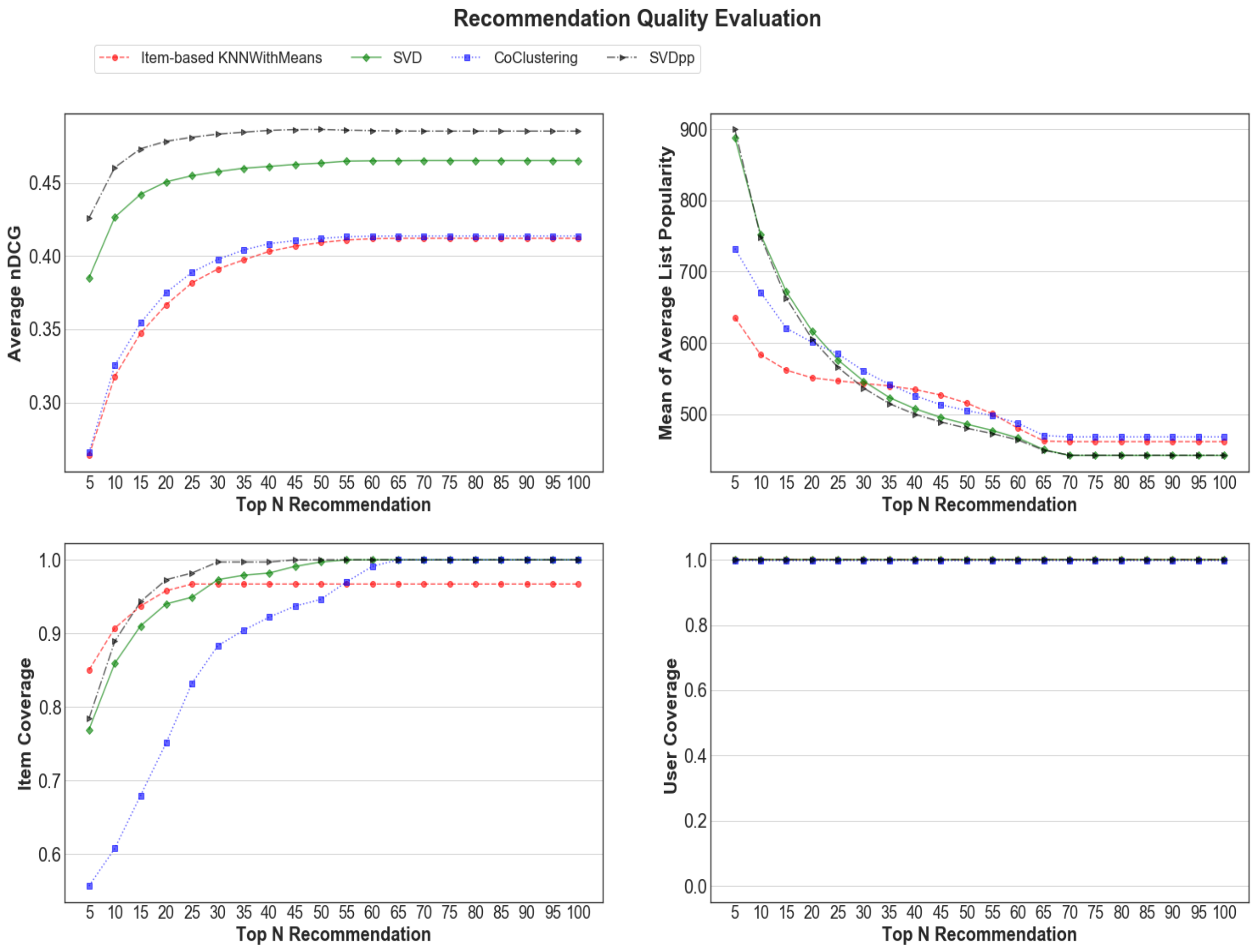
Sensitivity analysis for a hyperparameter of each algorithm was performed to show how a model's performance change based on the hyperparameter setting. For my experiment, it helped me to ensure that each algorithm was properly tuned from grid search to minimized unbiased RMSE. An example of sensitivity analyses for a hyperparameter for each algorithm is presented in the figure below.



Taking item-based KnnWithMeans algorithm as an example, the best testing RMSE was observed when neighborhood size was 5, which agree with the observation from cross validation. Moreover, since the testing RMSE was not larger than it's average RMSE of Cross Validation, the chance, the tuned KnnWithMeans is over-fitting, was very small. All these were evidence that the proposed hyperparameter setting from Grid Search is appropriate to minimize the unbiased RMSE. The same inference can be made from the figure for the other algorithms.

Evaluation Result

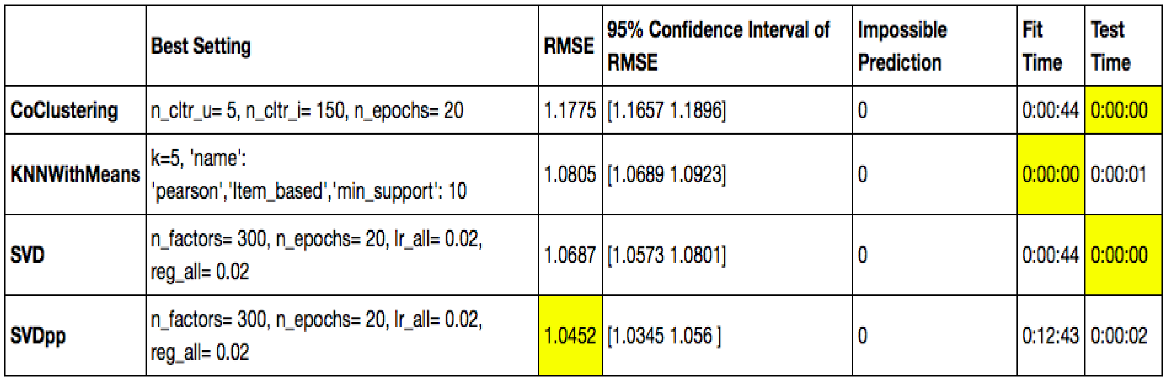
nDCG, Average List Popularity, and Coverage was used to evaluate each tuned algorithm's Top N Recommendation with relevance threshold set to 5. The results are presented in the following figures.



Lets take Top 10 Recommendations as our evaluation example. Based on the ranking metrics (nDCG), SVDpp (~47%) and SVD (~43%) produced better-ranked list of relevant anime for users than item based KNN (~32.3%) and CoClustering (~32.5). As for novelty metrics (Average List Popularity), measuring a system's ability to recommend long tail (unpopular) anime, KnnWithMeans outperforms others. Moreover, the user coverage and catalog coverage measure proportion of user or item covered and the result shows that all systems were able to make recommendation to all users but KnnWithMeans

(~91%) and SVDpp (~89%), SVD (~87%) were able to recommend significantly more anime than CoClustering (~57%).

RMSE are also computed as an aid for evaluations and it reveled that algorithms with better nDCG also tended to have better rating prediction accuracy in general. The results are presented in the following table.



	Best Setting	**RMSE**	**95% Confidence Interval of RMSE**	**Impossible Prediction**	**Fit Time**	**Test Time**
CoClustering	n_cltr_u= 5, n_cltr_i= 150, n_epochs= 20	1.1775	[1.1657 1.1896]	0	0:00:44	0:00:00
KNNWithMeans	k=5, 'name': 'pearson','Item_based','min_support': 10	1.0805	[1.0689 1.0923]	0	0:00:00	0:00:01
SVD	n_factors= 300, n_epochs= 20, lr_all= 0.02, reg_all= 0.02	1.0687	[1.0573 1.0801]	0	0:00:44	0:00:00
SVDpp	n_factors= 300, n_epochs= 20, lr_all= 0.02, reg_all= 0.02	1.0452	[1.0345 1.056]	0	0:12:43	0:00:02

SVDpp (~1.045) did best whereas CoClustering (~1.177) did worst on predicting rating. Noted that 95 CI of RMSE is calculated by using bootstrapping to get a better estimation of RMSE. Moreover, efficiently, KNNWithMeans is the fastest to fit and the SVDpp is the slowest to fit.

Challenges

The major challenge of completing this project was to get data and evaluation metric ready for a proper experiment. I spent lots of time on data preprocessing, evaluation metrics implementation, and data visualization. Eventually, I was able to overcome the challenge and present the more accurate and meaningful evaluation estimation in an easy to understand and analyze figures and tables.

Conclusion

About the Data and Algorithms

Based on the experiment, I observed that both prediction and recommendation performance of an algorithm are affected by many factors. Properly tuned hyperparameters and appropriately prepared data can improve an algorithm's performance and the evaluation accuracy. Moreover, defining a suitable recommendation task based on the properties of data and selecting the evaluation metrics accordingly can lead to more meaningful and accurate evaluation results.

From exploring the anime rating data, I learned that large parts of rating are in high rating area and only small parts of rating are in low rating area. It could mean that users has tendency to rate higher than average in general. Also, just like most of the data for recommender system, anime data exhibit 'long tail' distribution.

From evaluating the Top 10-Recommendation quality of each tuned algorithm, I learned that Matric Factorization Method (SVD, SVDpp) outperformed Neighborhood Method (Item-based KNN, CoClustering) on finding ranked list of relevant anime for users but has longer training time in general. However, if recommending unpopular (long tail) anime is important, KNNWithMeans is better at it than others. CoClustering, on the other hand, did poorly on most parts. It only has slightly better (almost the same) ranking accuracy than Item-based KNN but did not have the same strength as Item-based KNN on recommending long tail anime and covering up large proportion of anime. Moreover, fit time for CoClustering is just as long as SVD.

Limitations and Future Work

There were some limitations of the works. One of them was that the conclusion drawn from the experiment would only be valid for the anime rating data used for the study. Another one was that it was impossible to conduct evaluation on truly unknown ratings. In other words, there was no way to measure the user's satisfaction of the systems.

The future work to extend the project will then be evaluating algorithms on multiple dataset in multiple sizes in order to measure the scalability and generalization of the algorithms. If possible, another direction will be conducting an online evaluation with a/b testing to evaluate users’ satisfaction over different anime recommendation systems.