Hybrid Movie Recommender System

Introduction

"During the last few decades, with the rise of Youtube, Amazon, Netflix and many other such web services, recommender systems have taken more and more place in our lives. From e-commerce (suggest to buyers articles that could interest them) to online advertisement (suggest to users the right contents, matching their preferences), recommender systems are today unavoidable in our daily online journeys."
--<<Towards Data Science>>

In present times, a good recommendation system should be intelligent and personalized, so it can generate best results for different kinds of users. Our motivation is to implement a hybrid movie recommendation system that can predict movies that users may like accurately for users who have different profiles. An accurate recommendation system can help users find what they want which brings convenience to users in different ways. Meanwhile, for companies, a good movie recommendation system can increase revenue by bringing more customers, increasing average order value, and increasing view counts of movies. Our objective is to find popular algorithms of movie recommendation systems and compare their pros and cons. Then we will build a hybrid movie recommendation system that combines the results of different algorithms to improve accuracy of prediction for users with different backgrounds. Our approach is to use Movielens dataset as our database since the dataset contains all the information we need, such as movies, users, ratings, genres, and tags.

In this project, we will experiment with different types of recommender systems algorithms. By evaluating and comparing the accuracy of different models, find out the best combination of these models to create a high-precision hybrid movie recommendation system. For the methods categories covered in Fig.1, we are going to create algorithms to implement each of them.

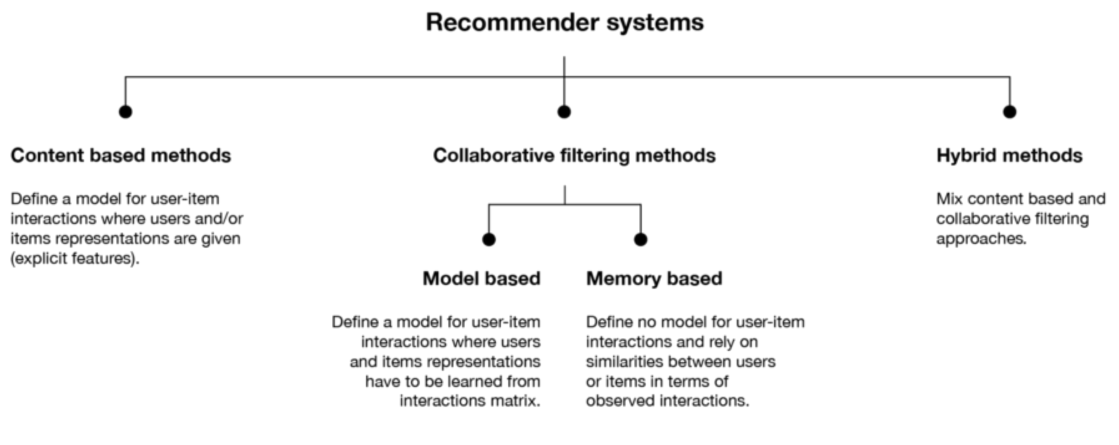

Recommender systems
Content based methods
Define a model for user-item interactions where users and/or items representations are given (explicit features).
Collaborative filtering methods
Model based
Define a model for user-item interactions where users and items representations have to be learned from interactions matrix.
Memory based
Define no model for user-item interactions and rely on similarities between users or items in terms of observed interactions.
Hybrid methods
Mix content based and collaborative filtering approaches.

Fig. 1.Summary of the different types of recommender systems algorithms.

The purpose of our recommender system is to recommend movies that users might like to watch. There are two major categories of methods : collaborative filtering methods and content based methods. We are going to implement a content-based algorithm manually since the dataset provides sufficient information and there are customized details that need to be added into our algorithm.
For collaborative filtering methods, we have more choices to do the implementation. By utilizing the Surprise library, we realize the matrix factorization algorithm (SVD and SVDpp) which is model based methods. Even the item-based K Nearest Neighbor algorithm didn't achieve high accuracy during validation, we still implemented it to compare to our manually item-item algorithm implementation.

System Design & Implementation

In order to choose the most accurate recommendation algorithm, we decided to implement all existing recommendation system algorithms.

1. Collaborative Filtering Methods

Collaborative methods for recommender systems are methods that are based on the past interactions recorded between users and items. We are going to use these past records of users to calculate similarities between users or movies and make predictions.Furthermore, collaborative filtering algorithms are divided into memory based and model based approaches.

- **Memory Based Approaches**
  Memory based approaches are what we usually call user-based or item-based(also known as item-item based) which will directly calculate the similarity between users or items and based on nearest neighbours search to suggest the most popular items among these neighbours. Since we are dealing with a giant dataset(users records over 20 million), it is extremely hard to apply a user-based algorithm. At the same time, if the user profiles have updates, the entire model needs to be recomputed to adapt new changes. Therefore, the computation is too expensive and our local RAM cannot afford it, so we decided to use an item-based algorithm. In the item-based approach, the system first finds similarity between all the item pairs. The similarity is calculated by using cosine similarity equation or pearson similarity equation. The similarity results are saved in a data frame for later use. In this way, the model does not need to rebuild the data frame everytime it makes predictions. It just finds the necessary information from the data frame. However, if there are updates of information of users or movies, the similarity data frame needs to be rebuilt. Then, the system makes predictions for a particular user. Based on the items that are rated by the user, the system finds the most similar items that are unrated by the user, and the system computes predicted rating for a particular item using a formula of weighted average. Finally, the system returns an item list that the user will presumably give high ratings. In this algorithm, we have two different implementations, one is using the python's Surprise library, and another one is a self implemented algorithm from scratch. For the self implemented part, a pivot table is built based on movies' identification and users' identification. A cosine similarity function from scikit-learn library is used to compute similarity between items. The algorithm finds the movies that are highly rated by the user, and it finds movies that are most similar to those highly rated movies. In those movies, the algorithm applies a weighted average equation with consideration of item similarities to find the unwatched movies that might get high ratings

from the same user. The results are the recommended movies. Although the computation of both implementations still takes a very long time, we get the ideal results.

- **Model Based Approaches**
  For model based collaborative approaches we utilize SVD and SVD++ algorithms of Surprise library.When baselines are not used, this is equivalent to Probabilistic Matrix Factorization. We choose these two algorithms because they show high accuracy in algorithms' validation. The SVD decomposes the sparse user-item matrix into a product of two smaller and dense matrices: a user-factor matrix that multiplies a factor-item matrix. The main idea of model based approaches is that it assumes an underlying "generative" model which can explain the user-item interactions and try to discover it in order to make new predictions.

2. Content Based Methods

Content-based methods are based on a description of the item and a profile of the user's preferences.These methods are best suited to situations where there is known data on an item (name, location, description, etc.), but not on the user. Content-based recommenders treat recommendation as a user-specific classification problem and learn a classifier for the user's likes and dislikes based on an item's features.

In this system, keywords are used to describe the items and a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past, or is examining in the present. It does not rely on a user sign-in mechanism to generate this often temporary profile. In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended.

Our implementation of content-based is a modified version, instead of creating item profiles and user profiles, calculating the similarities between user profiles and item profiles and at the end recommending the movies with highest similarities to the user, what we did is a little bit different but conceptually similar with the content-based method. What we did first is extract tags of a movie which have relevance higher than 0.8, after extracting the tags we combined the tags of each movie with the genres of each movie so here we have the item profile for each movie. The following steps are creation of user profiles and recommendations . After the creation of item profiles, we extract the movies each user likes the most, for example, the highest rating a user gives is 4, then we extract all the movies he/she rated 4 and add them to a list. We iterate through this list and get similar movies with weights(the higher the similarity, the higher the weight it has) from each movie in this list and add those similar movies to another list, from this list we can get the total weight of each movie, the movie with the highest total weight will be the no.1 recommended movie to the user, the movie with the second highest weight will be the no.2 recommended movie and so on.

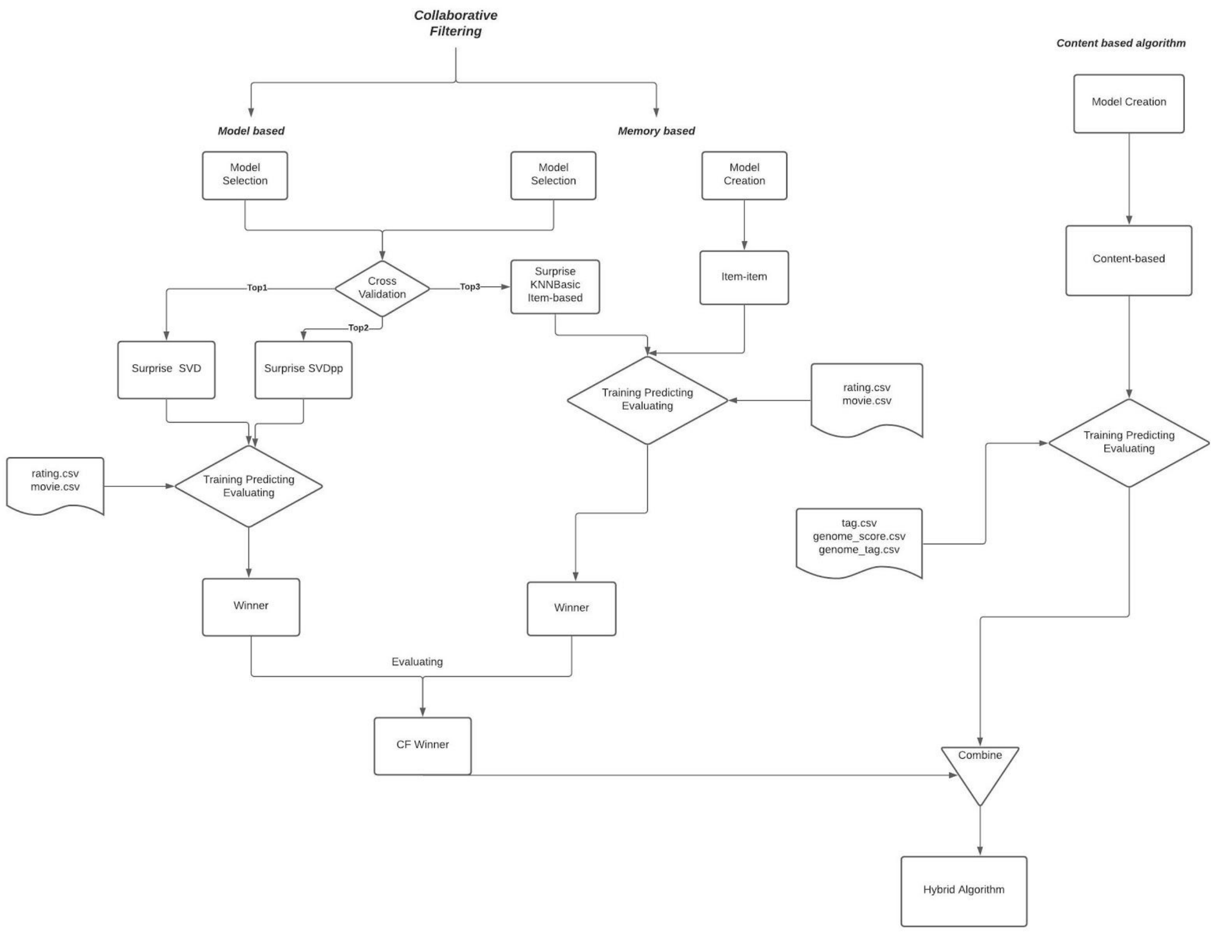


Fig.2 Project System Design

Experiments / Proof of Concept Evaluation

1.Data and Preprocessing

The dataset we used for this project is MovieLens 20M Dataset that is retrieved from Kaggle.com. The dataset contains 20000263 movie ratings and 465564 tag applications across 27278 movies. The data was collected from a total of 138493 users between January 09, 1995 and March 31,2015. All the data are stored in comma-separated value files. Based on the algorithms we used, the data that are related to ratings, movies, users, tags, and genres are used.

2. Methodology of Surprise Model Selection

By shuffling and splitting the user rating dataframes, we put randomly selected small dataset into every Surprise model. We utilize five fold cross validation and evaluate each algorithm by RMSE value. Fig.3 and Fig.4 shows the RMSE of each algorithm.

```
Algorithm          RMSE

KNN Basic          1.0412
KNN Means          1.064
KNN ZScore         1.065

SVD                1.0251
SVDpp              1.0223
NMF                1.0676

SlopeOne           1.0622
CoClustering       1.0687
```

Fig.3 The RMSE Value of Each Algorithm inSurprise

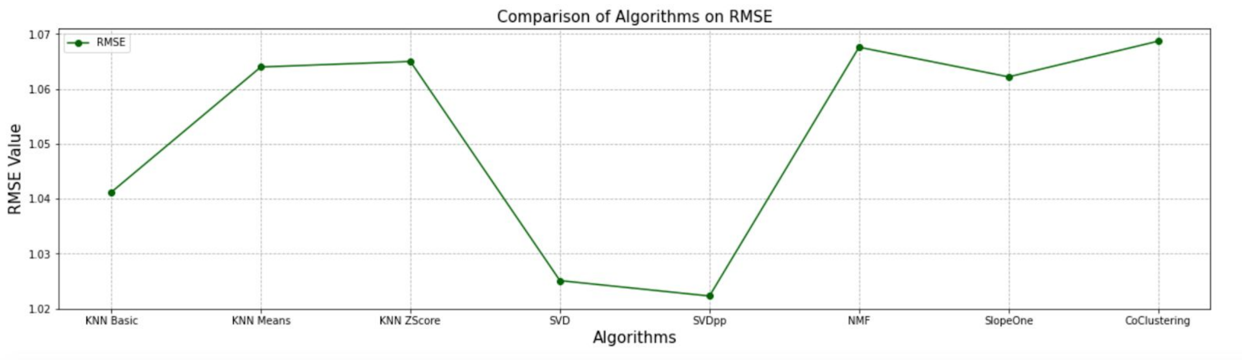


Fig.4 Comparison of Algorithm of RMSE

The SVD and SVD + + have better performance, so we select these two as our candidates. For hyperparameter tuning, we still use a small batch of data. We tried to use the whole set of data for tuning, the results are the same. And we utilize GridSearchCV to find out the best parameter combinations. The best parameter combination for SVD is {'lr_all': 0.01, 'n_epochs': 20, 'n_factors': 100, 'reg_all': 0.1}. The RMSE decreased from 0.963 to 0.9592. The best parameter combination for SVD++ is {'lr_all': 0.05, 'n_epochs': 30, 'n_factors': 150, 'reg_all': 0.2}. The RMSE changed from 1.0223 to 1.0229. After several rounds of tuning, the SVD++ didn’t show expected improvement, so we chose SVD as our final algorithm.
For the SVD model, we could use the entire dataset for model training, and we did, which takes a very long time and has the same performance as we use a relatively small size of data. And we try to use the same data set to train every winning algorithm, due to the difficulty of memory based algorithm computation, we chose 100,000 randomly picked data as training set. At the evaluation section, we are going to use RMSE and Precision at K to evaluate the SVD model. The evaluation test set is split from the whole dataset. The testset percentile is 20% which is around 4 million data.

3.Methodology of Item-item based Model

In the item-item based collaborative filtering part, the preprocessing step is to combine the data in rating and movie into a single dataframe. Then, the matrix of item to item is formed by using a pivot table, and missing values are filled with zeros. At the same time, since the size of the dataset is huge, optimization for types of variables are applied to reduce memory space taken in future processing. For instance, the

type “float 64” is converted to the type “float 32”. The optimization will save half of the original memory space, so more data can be fed into future steps of processing.
Root mean square error (RMSE) is used to evaluate the self implemented item-based collaborative filtering. It is a metric that is used to measure the differences between values predicted by model and actual values. During the evaluation, a small portion of the user data is used as test data to evaluate the performance of the model. In this step, a new function is created to calculate predicted ratings of test data. A function of mean square error from scikit-learn library is used to calculate the RMSE value. Several different test sets are used for evaluation. The RMSE values varys from 0.9 to 1.1. RMSE is always non-negative, and a lower RMSE value means smaller error of the prediction the model made. For a self implemented model, this RMSE value is normal and acceptable, although there are some noticeable errors in predictions.

4.Methodology of Content-based Model

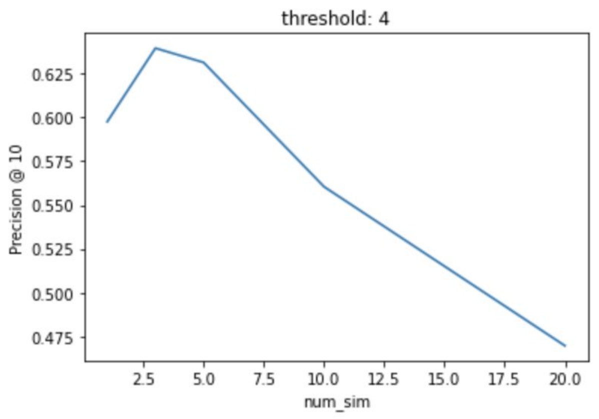
We are not able to use RMSE because unlike collaborative filtering, content-based methods or our modified content-based method didn’t estimate the rating for each movie so we use precision at 10. For the Top 10 recommendations, check what’s the percentage of good recommendations. For example, if there are 6 movies out of 10 recommendations the user has watched before and the ratings of 3 recommended movies are above threshold (threshold can be defined by anyone, we implemented using two different thresholds: 4 and mean rating of each user) then the precision for this user is 3/6 = 0.5. The overall model precision is the sum of values of precision of n users divided by n. We choose n=100 because of performance reasons (100 users are randomly selected).

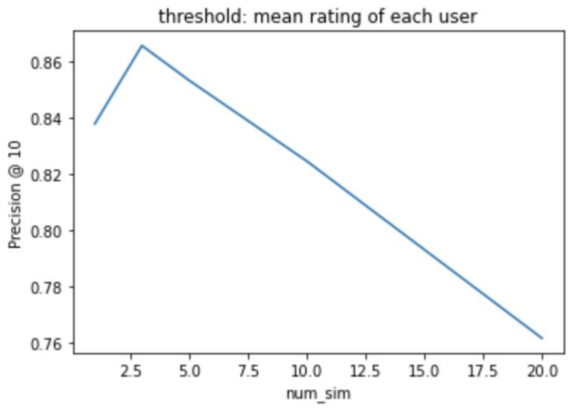
Results for modified content-based method:

Evaluation metric: Precision at 10
Number of user evaluated: 100

```
Threshold = 4:
number of similar movies = 20:  0.4697976190476191
number of similar movies = 10:  0.5604603174603174
number of similar movies = 5:  0.6313095238095239
number of similar movies = 3:  0.639452380952381
number of similar movies = 1:  0.5975555555555555


Threshold = mean rating of each user:
number of similar movies = 20:  0.7614880952380951
number of similar movies = 10:  0.8245714285714284
number of similar movies = 5:  0.8533095238095236
number of similar movies = 3:  0.8656428571428573
number of similar movies = 1:  0.8378333333333333
```





Analysis:

1. Best precision when num_sim = 3 for both threshold
   Num_sim stands for the number of similar movies returned when inputting a single movie.
   It's easy to see that when num_sim = 3 the model has the best precision at 10, after num_sim = 3, the precision starts to descend as num_sim increases, it seems that as we include more movies having smaller similarities, the model becomes less accurate. Another observation is that the best precision doesn't happen when num_sim = 1, it probably is because there are too few movies to reference.
2. We implement the precision metric at two different thresholds, the first one is threshold = 4, choosing 4 is because with a rating that scales from 1 to 5, 3 is considered "OK" and 4 is considered "good". However, not all user have the same rating criterion, maybe user A is a strict rater, the lowest rate he/she gives is 1 and the highest rate 3, on the other hand, user B is a easy rater, the lowest rate he/she gives is 3 and the highest 5, in this case user A will never have a movie that pass the threshold. To solve this problem, we implemented mean-centered rating which is a kind of normalization, that is to say, we set the threshold to the mean of ratings of a user. For example, the average rating of user A is 2.5, then a movie recommended to him/her which he/she rated 3 is considered a good recommendation. Obviously, we have better precision when using the second threshold.

5. Comparison of models

We compared the prediction results made from different models. One of the comparisons is the prediction coverage of different models. The information shows the percent of movies that different models are able to recommend.

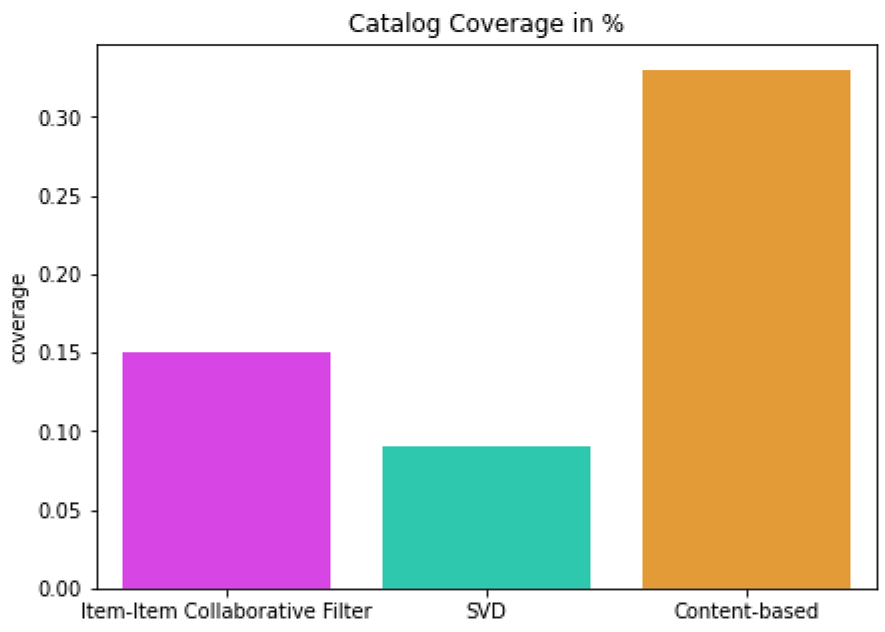


Fig.5 Predictions coverage of different models

Personalization is also measured between different models. Personalization is the dissimilarity between recommended movie lists of different models. Larger personalization score indicates greater dissimilarity between recommended movie list, and vice versa. Personalization score of our models' predictions is 0.62. Different algorithms do have significant differences on prediction results.

Discussion & Conclusions

1. Difficulties we are facing

In the project, one of the difficulties we had was the large size of the dataset. The size of data was too large, there were 20 million pieces of data. We could not fit all the data into our model with our own computers. We have tried to run our codes in other platforms, such as google cloud datalab and google colab and HPC, but we still could not solve the problem. Then, we used sampling methods to decrease the size of the dataset, so we could find ways to fit the majority of data to our models. Meanwhile, we did some optimization on variable types to allow our algorithms to take more data. For some instances, there was no need to keep the type "float 64" since there were not many decimals, so we converted those instances from the type "float64" to "float32". Similarly, some instances were converted from the type "int64" to the type "int32". By doing this, we saved half of the original memory space when processing those data. Therefore, we could fit more data to the models. In addition, for some algorithms, we processed data in chunks to handle large dataset. With all these combined strategies, we were able to fit 80 percent of the data to the models. Another difficulty was the evaluation of the model. Two of our algorithms were self implemented, so we could not apply existing libraries to evaluate the models. It was hard to write evaluation functions for the models. For the item-item based collaborative filtering part, only RMSE was used for the evaluation. In addition, we tried to implement other deeper evaluations, but we were not able to achieve that. The performance of self implemented evaluation algorithms did not meet our expectations. The runtime of the algorithms was very slow, so we could not feed too much data for evaluation. After we run a single model on one computer, it already uses the whole RAM, so for the final version of this ipython notebook, we use the same data preprocessing steps, but run our model separately(on different laptops).

2. Conclusion

Throughout the project, the SVD model showed unparalleled advantages in accuracy and running time. And by separately running the script, we successfully tune and train each candidate model (SVD, KNNBasic item-based, item-item, content-based).
Time performance and the ability of handling large data were the biggest problem throughout the entire project. In the future, we should learn PySpark and use it to build our models. PySpark is a framework that has in-memory operations, distributed processing, fault-tolerance data objects, and integration with machine learning. Its performance is 100 times faster than traditional Hadoop MapReduce. Using sparks can boost our models' performance dramatically. In addition, we should explore other distributed techniques for data science, so we can handle large amounts of data and process them efficiently.

3. Exploring

We try to create a hybrid model which combines the SVD model and the Content-based model. We considered to use two way to set up the recommendation model as following:

- Mixed: Recommendations from different recommenders are presented together to give the recommendation.
- Meta-level: One recommendation technique is applied and produces some sort of model, which is then the input used by the next technique.

For the mixed hybrid model, we could simply present the results of the content-base model and SVD model at the same time. We also can use meta-level methodology to create a more accurate model. We first implement the SVD model and use the results of recommendation as input or selection set of content-base model. There is more to explore, but due to the limitation of our computer(RAM crushed after running only one model), we couldn't successfully run it. But we can do more experiments in the future as long as we have the solution to the OOM problem.

Project Plan / Task Distribution

Task	Owner	Status
Surprise SVD + SVDpp	Katy	Complete
Item-Item	Yanzhou	Complete
Content-Based + Surprise item-based	Brian	Complete
Hybrid Algorithm	Katy	Complete

Evaluation And Comparison	Katy, Brian, Yanzhou	Complete
GIthub	Katy	Complete
Report	Katy, Brian, Yanzhou	Complete
Slides	Katy, Brian, Yanzhou	Complete
Code Reorganization	Katy	Complete

Reference

https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada