Movie Recommendation Report

Jing Dong, Maxine Xin, Ran Yan, Jojo Zhou

May 2, 2020

Contents

1 Abstract

We intended to build a generalized and generic recommendation system pipeline in which developers and researchers could apply most current algorithms easily while they are also provided with various easy-to-add evaluation metrics. In our system, we used various K-Nearest Neighbor and Matrix Factorization approaches. In addition, we pick one best-performed algorithm in KNN and one best-performed algorithm from matrix factorization to construct the hybrid algorithm. In our experiment, we found that tuned SVD and tuned Baseline KNN have the best overall performance. Baside the accuracy, KNN has high diversity and novelty compared to SVD. Therefore, we optimized SVD and Baseline KNN algorithms by tuning their parameters. We later combined them into the hybrid algorithm in an attempt to produce a more user-based recommendation system relevant to novelty and diversity.

2 Introduction

Today, the Internet is used in almost every aspect of our life. At the same time as all the tools and information are available and accessible to us, they can be overwhelming. Thus, many businesses and researchers, especially those in online-shopping, video streaming, and social network application industry, are exploring ways to make services and tools more personalized so that they could eliminate options for their users. The performance of the recommendation system is crucial for the success of business as it directly influences customer satisfaction (Wei et al., 2017).
In this report we want to build a hybrid movie recommendation system. The supporting algorithms for the hybrid recommender will be selected from collaborative filtering algorithms.
The collaborative filtering algorithm essentially recommends items based on user and item features. More precisely, we have a set of users U and a set of movies M. For each $u_i \in U$, u_i has a set of movies M_i that they have already seen and rated. The AI learned from $\bigcup u_i$ and $\bigcup M_i$, based on which recommends a set of movies $R_i \subseteq M$ for each user u_i.
We will choose collaborative filtering algorithms from two main categories: the Matrix Factorization algorithms and K-Nearest Neighbors algorithms. Matrix Factorization algorithms considers both user features and movie features evenly. While movie features describe attributes, like being romantic, being horrific and so on, user features describe how much the user likes these attributes. Matrix Factorization will then predict ratings based on user and movie feature interaction. On the other hand, in this project we applied user-based KNN algorithms which focuses on user features. The KNN algorithms will mainly focus on the similarity between users and recommend movies based on most similar users.
We will then construct the hybrid movie recommendation system from these algorithms. We are interested in doing so since hybrid system is supposed to perform better than its components. Moreover, since every algorithm has its pros and cons, we can weight each algorithm to exaggerate user preferences.

3 Survey of Related Work

Generally speaking, two common approaches investigated in previous research are content filtering and users' past activity, or collaborative filtering (Koren, Bell and Volinsky, 2009). The former creates a profile for each product based on its internal characteristic or feature and then matches the user with a related product. One challenge is the difficulty of collecting all the information of the "content". Collaborative filtering, on the other hand, "analyzes the relationships between users and interdependencies among products to identify new user-item associations" (Koren, Bell and Volinsky, 2009). It generally performs better than content filtering and it doesn't share the same problem that content filtering approach has. However, it has a problem known as cold start: at the beginning of building the system, as there is not much information on the product and user, it does not function well. Thus, in our project, we want to design and build a recommendation system using collaborative filtering with different implementations. If we have time, we would also add a content-based approach so that we could compare their performance ourselves and combine them for a hybrid recommendation system.

For matrix factorization, Singular Value Decomposition (SVD) is widely used for identifying latent semantic factors with a stochastic gradient descent optimization (Funk, 2006). One problem with SVD is that it requires the user-rating matrix, which might be a sparse dataset (Koren, Bell and Volinsky, 2009). It won't be a problem for our case, as we have a dataset with relatively sufficient user-rating data. There are also various algorithms based on SVD like SVD++, Probabilistic Matrix Factorization (PMF) (Salakhutdinov and Mnih, 2008), and Non-negative Matrix Factorization (NMF) (Lee and Seung, 2001). On the other side, we also applied KNN based algorithms. The KNN classification method has been widely applied in recommendation systems. For instance, Adeniyi, Wei and Yongquan applied KNN to develop an automated web usage data mining and recommendation system (Adeniyi, Wei and Yongquan, 2014). They use KNN algorithm to identify visitors and recommend browsing options that meet the needs of the specific user in Real-Time. They chose KNN over other machine learning techniques since it is consistent with relatively smaller errors. It is also preferred for its short running time which is necessary for the Real-Time usage. KNN algorithm has also been applied to Text and Document Mining. It is used to classify a set of text documents into different categories and then return the most relevant documents(Bijalwan, Kumar, Kumari and Pascual, 2014). In specific, it constructs vectors for documents, makes the centroid vector for each class, calculates similarity between document vectors and class vectors, and finally classifies the document to class with maximum similarity.

Lastly, there are also researches done on developing hybrid recommendation systems which use more than one filtering algorithms to overcome weakness of one specific algorithm. From the taxonomy presented by Burke, there are generally seven classes of hybrid recommendation systems: weighted, switching, mixed, feature combination, feature augmentation, cascade, meta-level (Burke, 2002). Based on the survey of Thorat, Goudar and Barve (2015), generally speaking, hybrid systems can be implemented in four ways. The approach we took for our system is to implement different collaborative filtering individually and aggregate their predictions by a given weight.

Different from the previous works, we intended to build a generalized and generic recommendation system pipeline in which developers and researchers could apply most current algorithms or a hybrid version of algorithms easily while they are also provided with various easy-to-add evaluation metrics including hit rate, cumulative hit rate, rating hit rate, average reciprocal hit rank, diversity, novelty, user coverage (more details are in Section 6). There are also tuned examples in our system. As our system provides many metrics, it could give researchers a more holistic view of the performance of their model. Additionally, in our hybrid algorithm, we pick one best-performed algorithm in KNN and the other from Matrix Factorization. Since the accuracy is similar based on our experiment results but there are differences on coverage, diversity and novelty, we build one hybrid algorithm. The hybrid algorithm is for the users who love to explore various types of movies and some unpopular movies. This improvement is expected to be more user-friendly.

4 Formulation

We formulate this problem as an optimization task. Therefore, we will use collaborative filtering to predict the rating user$_i$ gives to movie$_j$ that optimize assumptions of dependencies between users and movies. Collaborative filtering algorithm infers ratings only based on the user's past rating patterns. In addition, it considers the dimensions of human tastes and movie quality which a content-based filtering algorithm could not analyze. We also prefer collaborative filtering over content-based filtering based on the fact that collaborative filtering performs well when sufficient data is available. The problem space is formulated as a user-rating matrix and our task is to fill in the empty cells in which the user$_i$ gives rating to movie$_j$.

In specific, we choose the matrix factorization algorithm and k nearest neighbors algorithm as models that will be used as inputs to the hybrid model.

4.1 Matrix Factorization algorithm

Matrix factorization algorithm represents the rating matrix as the factorization of **user feature matrix** and **latent movie feature matrix**.

It obtains the following information from the dataset.

- U: a set of n users
- M: a set of s movies
- F: a set of t features selected by us
- M: an $n \times s$ matrix
  - $W[i, j] = 1$ if user $u_i \in U$ has watched movie $m_j \in M$
  - $W[i, j] = 0$ if otherwise
- R: an $n \times s$ matrix, $R[i, j]$ =ratings from user $u_i \in U$ to movie $m_j \in M$, if $W[i, j] = 1$.

We have 2 parameter matrices A, B, whose entries are randomly filled in at the beginning:

- A: $n \times t$, $A[i, k]$ represents how much user $u_i \in U$ likes feature $f_k \in F$

- B: $t \times s$, each $B[k, j]$ represents how relevant feature $f_k \in F$ is to the movie $m_j \in M$.

We can use the entries of the parameter matrices to express the predicted rating, $p(i, j)$, of a user u_i to a movie m_j. In words, the predicted rating depends on how much the user likes each feature, and how relevant the movie is to each feature.

$$p(i, j) = \sum_{k=1}^{t} A[i, k]B[k, j].$$

Now we've discussed the set-up of our problem, we can start to talk about the procedure and algorithm we use: For each $i, j, W[i, j] = 1$, we compute the error between $p(i, j)$ and $r(i, j)$, and apply the gradient descent algorithms to find the optimal values to fill in to entries of A, B. During this process of tuning the parameters, we also include regularization to prevent over fitting. After the tuning stops, we will have two good enough parameter matrices, which in turn will yield good enough predicted ratings. Now for each user u_i, we will recommend movies m_j where $W[i, j] = 0$ and $p(i, j)$ is high.

4.2 K nearest neighbors algorithm

K nearest neighbor algorithm identifies k closest example in the feature space and classifies the object by the majority vote. In specific, we will use a user-based k nearest neighbor algorithm to predict unrated movies scores based on similar user preferences.
In this scenario, we are given the training data $(\vec{x_1}\ , \vec{y_1}\ ), \cdots, (\vec{x_n}\ , \vec{y_n}\ )$, where $\vec{x_i} \in X$ is the feature vector of user$_i \in U$ and $\vec{y_i} \in \{0, 1, 2, 3, 4, 5\}$ is the rating that user$_i$ gives to movie$_i \in M$. We will use cosine similarity function:

$$sim(u, v) = \frac{\sum_{i \in I_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I_{uv}} r^2{}_{ui}} \sqrt{\sum_{i \in I_{uv}} r^2{}_{vi}}}$$

which calculates the angle between user vectors u, v in the feature space.
The rating of user u giving to moive$_i$ is calculated by summing the ratings of all k similar users v to moive$_i$ weighted by similarities between users u, v. The formula is as follows:

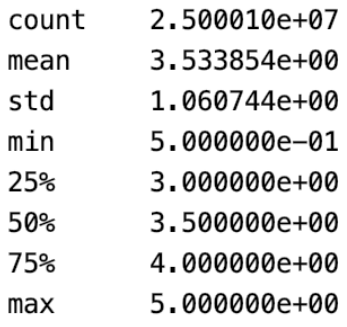
$$\hat{r}_{ui} = \frac{\sum_{v \in N_i{}^k u} sim(u, v) r_{vi}}{\sum_{v \in N_i{}^k u} sim(u, v)}$$

where $N_i{}^k u$ is the set of k most similar users of u, r_{vi} is the rating of user v giving to movie$_i$ and sim(u, v) is the similarity between users u and v.

5 Approach

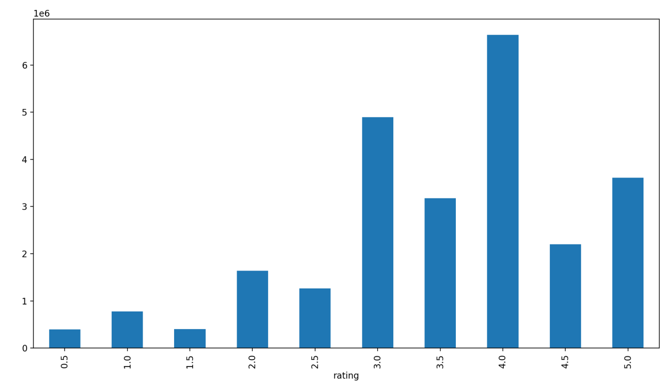
5.1 Data Set

In this final project, our dataset is ml-latest-small data set. It includes 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users. We mainly use two files: "movies.csv" and "ratings.csv". 100,000 data points is enough to make rational predictions and do not challenge the ability of the CPU for the laptop. Our main full data set is from "ratings.csv" which includes movieId, userId rating and timestamp. To make the TopN recommendations, we use "movies.csv" to get the name of recommended movies.



```
count    2.500010e+07
mean     3.533854e+00
std      1.060744e+00
min      5.000000e-01
25%      3.000000e+00
50%      3.500000e+00
75%      4.000000e+00
max      5.000000e+00
```

In this data set, the mean rating score is approximately 3.5, with minimum score of 0.5 and maximum score of 5. It fits to a skewed bell distribution, as shown in the histogram:



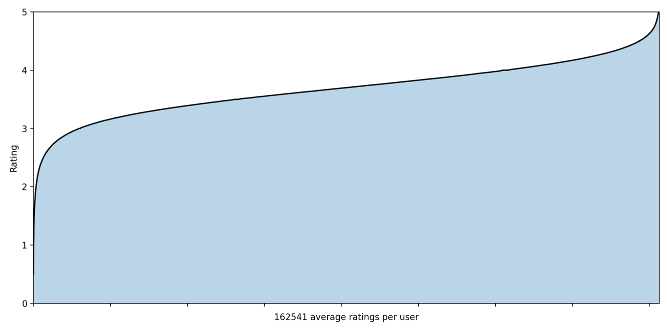


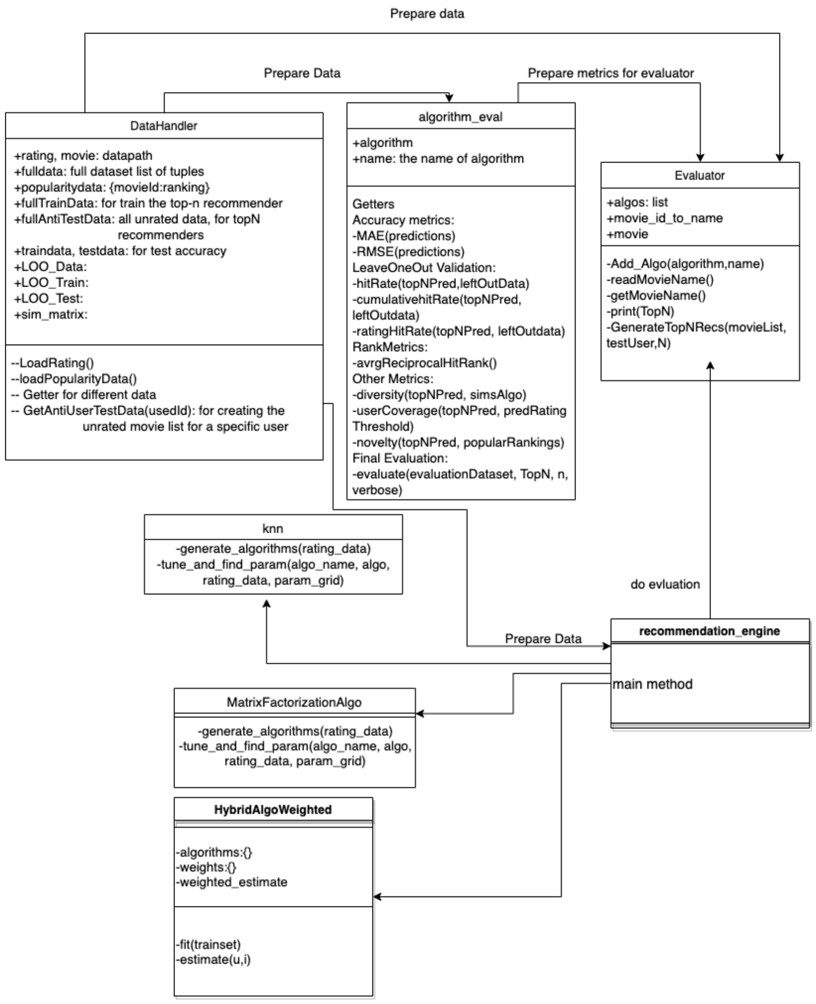
Figure 3 shows an overall rating trend: while there are only few people rate low consistently, there are also few people who rate consistently high.

5.2 Library and Language

The collaborative filtering, tuning and evaluation algorithms come from the library Surprise. We prefer this library since it is designed to build and analyze recommender systems that deal with explicit rating data. The programming language we use is python. We also use matplotlib to produce graphics.

5.3 Architecture

Here is the class-design in UML form.


Prepare data
Prepare Data
Prepare metrics for evaluator
DataHandler
+rating, movie: datapath
+fulldata: full dataset list of tuples
+popularitydata: {movieId:ranking}
+fullTrainData: for train the top-n recommender
+fullAntiTestData: all unrated data, for topN recommenders
+traindata, testdata: for test accuracy
+LOO_Data:
+LOO_Train:
+LOO_Test:
+sim_matrix:
--LoadRating()
--loadPopularityData()
-- Getter for different data
-- GetAntiUserTestData(usedId): for creating the unrated movie list for a specific user
algorithm_eval
+algorithm
+name: the name of algorithm
Getters
Accuracy metrics:
-MAE(predictions)
-RMSE(predictions)
LeaveOneOut Validation:
-hitRate(topNPred,leftOutData)
-cumulativehitRate(topNPred, leftOutdata)
-ratingHitRate(topNPred, leftOutdata)
RankMetrics:
-avrgReciprocalHitRank()
Other Metrics:
-diversity(topNPred, simsAlgo)
-userCoverage(topNPred, predRating Threshold)
-novelty(topNPred, popularRankings)
Final Evaluation:
-evaluate(evaluationDataset, TopN, n, verbose)
Evaluator
+algos: list
+movie_id_to_name
+movie
-Add_Algo(algorithm,name)
-readMovieName()
-getMovieName()
-print(TopN)
-GenerateTopNRecs(movieList, testUser,N)
knn
-generate_algorithms(rating_data)
-tune_and_find_param(algo_name, algo, rating_data, param_grid)
do evluation
Prepare Data
recommendation_engine
main method
MatrixFactorizationAlgo
-generate_algorithms(rating_data)
-tune_and_find_param(algo_name, algo, rating_data, param_grid)
HybridAlgoWeighted
-algorithms:{}
-weights:{}
-weighted_estimate
-fit(trainset)
-estimate(u,i)

Our framework design is inspired by the online Linked In Learning course– “Building Recommender Systems with Machine Learning and AI”.

For our recommender engine, we can divide it into two parts: evaluator and recommender.

For the evaluator, it is used for evaluating the recommendation algorithms in several dimensions: accuracy, leave-one-out validation, decision support and rank metrics.

To be more specific we first generate the algorithm eval class. In this class, our member variable is the name of the algorithm and the algorithm itself. Our evaluations metrics are divided into four dimensions: accuracy, LeaveOneOut validation, Rank metrics and other metrics like (diversity, coverage and novelty). After defining each metrics, we create the evaluate function storing the experimental result of each metrics into a dictionary. We also implemented other metrics like precision and recall, but we didn’t have time to fully test them so we didn’t include them in the final system.To be more specific, the reason we use Leave One Out validation instead of KFold validation is because of the limitation of our computing power. K Fold gives us too much pressure but do not make a big difference so we pick LeaveOneOut validation.

Next we generate the necessary dataset for our recommender. The dataset can also be divided into two groups : the raw dataset and the training/testing dataset. Thus, the full data is directly read from the file, the popularitydata is a dictionary whose key is movieId and the value is the rank of that movie. The other group is based on the surprise library, we deal with data to do the evaluation. The fullTrainData is with no fold predefined, which is the whole dataset that is used to train the model. The fullAntiTestData is all unrated movies, we prefilled in with global mean. The second part of this group is data used for testing accuracy. We split the full dataset into 80% of train data and 20% of test data. The third part of the data is for leave-one-out validation. The reason we choose to use leave-one-out validation instead of k-foldi is because our computing power can not take that expensive validation method. LOO_Data is a tuple of (trainset, testset) for Leave-one-out. LOO_Train is the training set for loo validation and LOO_Test is the test set for loo validation.The last part of the data is sim_matrix, which returns the similarity between each movie. In DataHandler, we load the data and create the necessary dataset. Then, we create getters.

Then we create the evaluator. The member variable is a list of stored algorithms we try to compare, a dictionary of movie_id_to_name whose key is movieid and value is its movie name and the path for the file. In this class, we are able to add the algorithm to the list. readMovieName() is used for loading the movie file and creating the dictionary for movie_id_to_name. The print() method is used for printing the whole result to the console and the GenerateTopNRecs method is used for generating the top N recommendations our engine made and printing it.

The second part is several algorithms we try to use. The first approach is Matrix Factorization and the second approach is KNN. We build classes for each approach and add different algorithms inside the classes. In HybridAlgoWeighted algorithm, we try to pick the best algorithm of KNN and the best algorithm of Matrix Factorization and we weighted their results to make recommendations.

The recommendation engine is our main class.

6 Experiments Analysis:

6.1 Initial Models

6.1.1 Matrix Factorization

The initial models of matrix factorization algorithm includes SVD, PMF, SVD++, and NMF.
SVD associates each movie i to a movie-feature vector $q_i \in \mathbb{R}^f$ and associates each user u a user-feature vector $p_u \in \mathbb{R}$. By construction, the rating user u giving to movie i is the inner product of the movie-feature vector and user-feature vector, as shown below:

$$\hat{r}_{ui} = \mu + b_u + b_i + {q_i}^T p_u$$

where μ is the global average rating, b_i is the item bias, b_u is the user bias, and ${q_i}^T p_u$ stands for user-item interaction. To estimate the feature vectors p_u and q_i, as well as b_u and b_i, SVD minimizes the regularized squared error on the set of known ratings:

$$\sum_{r_{ui} \in R_{\text{train}}} (\hat{r}_{ui} - r_{ui})^2 + ({b_i}^2 + {b_u}^2 + ||q_i||^2 + ||p_u||^2)$$

and the minimization is performed by stochastic gradient descent.
The probability matrix factorization(PMF) for surprise simply puts down the bias and set $\hat{r}_{ui} = {q_i}^T p_u$. SVD++ takes into account user's implicit feedback to predict ratings. When explicit feedback like rating is not available, SVD++ will analyzes user's implicit feedback like purchase history, browsing history and search pattern to predict the rating. Lastly, non-negative matrix factorization is very similar to SVD, where prediction $\hat{r}$ is set as $\hat{r}_{ui} = {q_i}^T p_u$, while q_i and p_u are kept positive.

Name	RMSE	MAE	HR	CHR	ARHR	Diversity	Coverage	Novelty	RHR
Random	1.419	1.135	0.069	0.068	0.0	0.051	50.0	875.643	Rating HitRate 1.0 0.154 2.0 0.079 2.5 0.067 3.0 0.086 3.5 0.082 4.0 0.048 4.5 0.042 5.0 0.083
SVD	0.871	0.671	0.085	0.098	0.0	0.039	39.889	451.421	Rating HitRate 3.0 0.031 3.5 0.049 4.0 0.086 4.5 0.167 5.0 0.193
PMF	0.976	0.752	0.102	0.114	0.0	0.048	32.913	444.789	Rating HitRate 2.0 0.026 3.0 0.039 3.5 0.016 4.0 0.112 4.5 0.188 5.0 0.229
SVD++	0.858	0.66	0.095	0.107	0.0	0.05	39.705	554.687	Rating HitRate 2.0 0.026 3.0 0.047 3.5 0.033 4.0 0.102 4.5 0.167 5.0 0.202
NMF	0.918	0.705	0.09	0.099	0.0	0.127	48.457	1043.008	Rating HitRate 2.0 0.053 3.0 0.055 3.5 0.049 4.0 0.086 4.5 0.167 5.0 0.174

Analysis: From the experiment, we found that SVD++ and SVD showed the highest accuracy. Then we tune both SVD and SVD++, because of the limitation of the computing power, only the SVD show the result. Thus, we choose to pick tuned SVD as part of our hybrid algorithm. From the data, we can see that random as the worst performance which makes sense as a baseline model. PMF has the worst performance overall. Since PMF is the unbaised verison of SVD, we could infer that user bias takes an important role in ratings. NMF has the second worst performance and is slightly better than PMF, which means that having negative elements in feature vector makes the result worse. Now we dive deep into different dimensions of SVD and SVD++.

From the data, we can see that the hit rates data are not really high, but that could be due to the large scale of our dataset. Also, although with high accuracy performance, the coverage and novelty of SVD and SVD++ are actually below the Random. However, these lower numbers do not necessarily reflect bad performance of the algorithms. If a user likes to view more diverse and new things, then SVD and SVD++ might be a little sub-optimal in this case. On the other hand, if a user wants to stay safe with the popular or familiar stuffs, then a low diversity and novelty rating might be actually good for that user. Hence, depending on the scenarios, SVD and SVD++ are two algorithm with quite ideal performances, especially for users with low need for novel recommendations.

6.1.2 K Nearest Neighbors

The initial models of k nearest neighbors include KNNBasic, KNNWithMeans, KNNWithZscore, and KNNBaseline. KNNBasic is the original k nearest neighbor algorithm that only considers k most similar user ratings with respect to similarity between users, as shown in the formula:

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i{}^k{}u} sim(u,v) r_{vi}}{\sum_{v \in N_i{}^k{}u} sim(u,v)}$$

KNNWithMeans takes individual rating habits into account. Since there are people who like to rate very low or very high consistently, individuals either being strict or easy to ratings may influence the experiment outcome. Therefore, KNNWithMeans adds user mean rating score to the formula, as shown below:

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in N_i{}^k u} sim(u,v)(r_{vi} - \mu_v)}{\sum_{v \in N_i{}^k u} sim(u,v)}$$

where μ_u and μ_v are mean rating scores of users u and v. KNNWithZscore in addition normalizes KNNWithMeans formula, as shown below:

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in N_i{}^k u} sim(u,v)(r_{vi} - \mu_v)}{\sum_{v \in N_i{}^k u} sim(u,v)}$$

where σ_u and σ_v are standard deviations of rating scores of users u and v. Lastly, KNNBaseline takes baseline rating into consideration, the formula is shown as below:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{v \in N_i{}^k u} sim(u,v)(r_{vi} - b_{vi})}{\sum_{v \in N_i{}^k u} sim(u,v)}$$

Results

Name	RMSE	MAE	HR	CHR	ARHR	Diversity	Coverage	Novelty	RHR	
									Rating	HitRate
Random	1.419	1.135	0.069	0.068	0.0	0.051	50.0	875.643	1.0	0.154
									2.0	0.079
									2.5	0.067
									3.0	0.086
									3.5	0.082
									4.0	0.048
									4.5	0.042
									5.0	0.083
BasicKNN	0.972	0.747	0.111	0.125	0.0	0.089	50.0	781.518	2.0	0.073
									2.5	0.048
									3.0	0.053
									3.5	0.053
									4.0	0.152
									4.5	0.179
									5.0	0.18
WithMeanKNN	0.901	0.689	0.062	0.072	0.0	0.276	49.446	2002.936	2.0	0.024
									3.0	0.023
									3.5	0.053
									4.0	0.076
									4.5	0.077
									5.0	0.125
WithZScoreKNN	0.898	0.682	0.07	0.078	0.0	0.252	49.575	1861.054	2.0	0.073
									3.0	0.03
									3.5	0.053
									4.0	0.082
									4.5	0.103
									5.0	0.125
BaseLineKNN	0.878	0.672	0.085	0.097	0.0	0.22	49.659	1566.796	2.0	0.049
									3.0	0.03
									3.5	0.053
									4.0	0.114
									4.5	0.154
									5.0	0.148

We could see from the result that the base line KNN had the best accuracy performance among all the KNN algorithms. Baseline KNN takes how much an user u and item i deviates from the average into account. Therefore, we can infer that the user and item deviations are important to performance accuracy. This is confirmed by the performance of WithZScoreKNN, which is the second best and which normalizes the personal difference. This is consistent to the result from Matric Fatorization experiment, in which personal bias is important to accuracy. It is as expected the rest of algorithms perform from good to bad are WithMeanKNN, BasicKNN, and Random, which take personal bias less and less into account.
Hence we chose the BaselineKNN as the one that we would tune to be included in our hybrid approach. One thing worth noticing is that, to the contrary of SVD and SVD++, BaselineKNN actually has really high Diversity and Novelty data. Hence, if a user has a high need for novel and diverse recommendations, BaselineKNN would be a really good choice with high accuracy.

6.2 Optimization Tuning

Our design logic is we find the best model for KNN and Matrix Factorization and then tune the best one for each approach. After we get the optimal tuning for the one best algorithm in each KNN and MF approach. We used the surprise.CVGridSearch package for the tuning process. Grid search performs exhaustive searching through the hyperparameter space. In specific, the surprise.CVGridSearch algorithm computes accuracy metrics for an algorithm on different parameters measured cross-validation. We chose to test different sets of parameters for k nearest neighbors and matrix factorization algorithms.

6.2.1 Tuning for K Nearest Neighbors

- Parameter: k: The (max) number of neighbors to take into account for aggregation
- Result: We choose the KNNBaseline as the best model d then we tune the KNNBaseline with GridSearch. Based on the majority vote principle, the best parameter for KNNBaseline is $k = 30$.

Results, KNN-tuned

Name	RMSE	MAE	HR	CHR	ARHR	Diversity	Coverage	Novelty	RHR	
									Rating	HitRate
blKNN_tuned	0.873	0.667	0.085	0.099	0.0	0.208	49.685	1499.049	2.0	0.024
									2.5	0.037
									3.0	0.045
									3.5	0.062
									4.0	0.087
									4.5	0.115
									5.0	0.177

6.2.2 Tuning for Matrix Factorization

- The parameter we choose to tune are:
  - n_factors: the number of factors / features we choose
  - n_epoch: the number of iteration of the stochastic gradient descent process
  - lr_all: the learning rate for all parameters, which also serves as an implicit regularizer
- There also exist two other parameters init_mean, and init_std for the normal distribution, that people have commonly tuned for Matrix Factorization, but aftering tuning those two parameters for our smaller test dataset, the result was not significantly affected, hence we did not include init_mean and init_std in the tuning for our final dataset, also for the purpose of running time saving.

Results, SVD-tuned

Name	RMSE	MAE	HR	CHR	ARHR	Diversity	Coverage	Novelty	RHR	
									Rating	HitRate
SVD_tuned	0.853	0.656	0.085	0.098	0.0	0.076	45.767	713.934	0.5	0.1
									2.0	0.026
									3.0	0.034
									3.5	0.073
									4.0	0.078
									4.5	0.13
									5.0	0.19

6.3 Hybrid

We chose one tuned algorithm from each KNN and Matrix Factorization to form our hybrid algorithm. We chose KNN baseline and SVD to tune, as they had the best accuracy performance. As discussed earlier, these two algorithms have opposite performance in regards to diversity and novelty, hence we could be able to adjust the weight assigned to each algorithm based on how much a user prefer novel and diverse recommendations. If a user has a high preference towards novelty and diversity, we would be able to assign a high weight to KNN. On the other hand if a user does not like novel and diverse recommendations, we would assign a high weight to SVD. We illustrated this approach by setting the preference of the user towards diversity and novelty as 0.8. We have results listed in the following:

Results (0.8 KNN, 0.2 SVD, both tuned)

Name	RMSE	MAE	HR	CHR	ARHR	Diversity	Coverage	Novelty	RHR	
									Rating	HitRate
hybrid	1.822	1.497	0.102	0.102	0.0	0.043	50.0	687.362	0.5	0.1
									1.0	0.077
									1.5	0.143
									2.0	0.154
									3.0	0.103
									3.5	0.055
									4.0	0.114
									4.5	0.074
									5.0	0.121

From the result, unfortunately we ended up with a relatively low accuracy performance. It seemed that the weight assigned to each algorithm still needed further more thorough adjustments, which we will discuss more in the conclusion seciton.

7 Conclusions

From the results, we concluded that the KNN baseline and the SVD, SVD++ had the best accuracy performance. KNN baseline had high diversity and novelty performance, while SVD, SVD++ had low diversity and novelty performance. We attempted to find a way to adjust the diversity and novelty performance of the prediction algorithms by combining them into a hybrid approach. However, unfortunately such attempt did not gave us the ideal results, which pointed to us a new direction to explore on. In the future, one important direction to explore one would be to find a way to adjust the weight assigned to each algorithms in the hybrid approach to match the preference of the user towards novelty and diversity, meanwhile maintaining the high accuracy performance of the prediction algorithms.

8 References

1. Adeniyi, D.A. Wai, Zune Yongquan, Y.. (2014). Automated Web Usage Data Mining and Recommendation System using K-Nearest Neighbor (KNN) Classification Method. Applied Computing and Informatics. 36. 10.1016/j.aci.2014.10.001.

2. Bennett, James and Stan Lanning. “The Netflix Prize.”, 2007.

3. Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. 2001. URL: http://papers.nips.cc/paper/1861-algorithms-for-non-negative-matrix-factorization.pdf.

4. Funk, Simon, “Netflix Update: Try This at Home.” Accessed May 2, 2020. https://sifter.org/ simon/journal/20061211.html.

5. James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. 2010. The YouTube video recommendation system. In Proceedings of the fourth ACM conference on Recommender systems (RecSys ’10). Association for Computing Machinery, New York, NY, USA, 293–296. DOI:https://doi.org/10.1145/1864708.1864770

6. Kane Frank, LinkedIn. “Building Recommender Systems with Machine Learning and AI - Deep Learning Introduction.” Accessed May 3, 2020. https://www.linkedin.com/learning/building-recommender-systems-with-machine-learning-and-ai/deep-learning-introduction.

7. Wei, Jian, Jianhua He, Kai Chen, Yi Zhou, and Zuoyin Tang. “Collaborative Filtering and Deep Learning Based Recommendation System for Cold Start Items.” Expert Systems with Applications 69 (March 2017): 29–39. https://doi.org/10.1016/j.eswa.2016.09.040.

8. R. Burke, “Hybrid recommender systems: survey and experiments”, User Modeling and User-Adapted Interaction 12 (4) (2002) 331–370.

9. Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. 2008. URL: http://papers.nips.cc/paper/3208-probabilistic-matrix-factorization.pdf.

10. Thorat, Poonam B., R. M. Goudar and Sunita Barve. "Survey on Collaborative Filtering, Content-based Filtering and Hybrid Recommendation System." International Journal of Computer Applications 110 (2015): 31-36.

11. Y. Koren, R. Bell and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," in Computer, vol. 42, no. 8, pp. 30-37, Aug. 2009.

12. Zhao, Changwei; Sun, Suhuan; Han, Linqian; Peng, Qinke (2016). "Hybrid Matrix Factorization for Recommender Systems in Social Networks". Neural Network World. 26 (6): 559–569. doi:10.14311/NNW.2016.26.032.

13. Bijalwan, V., Kumar, V., Kumari, P., Pascual, J. (2014). KNN based machine learning approach for text and document mining. International Journal of Database Theory and Application, 7(1), 61-70.

9 Appendix

From the proposal to the design implementation of our system, and to the experiment final report, we divided the work pretty much equally. Every group member is very responsible for their part!! :D