Recommendation System for Amazon Apps

Pin-Hao Chen (phc2121), Leshen Sui(ls3452), Siwen Tang (st3101), Jingyi Wang (jw3562)

1. Introduction

1.1. Background

This project aims to build a recommendation system for Amazon apps. Nowadays, an increasing number of online companies are taking advantages of recommendation engines to attract user's attention and enrich shopping potential. Over the last 4-5 years, use cases of recommender system have been expanding rapidly across many fields, and this trend is expected to continue. Everyday people receive messages and emails about product or service they might be interested in, therefore they will feel easier to purchase the right product. Companies also make personalized recommendation to spread out good product and increase the profit. The popularity and usefulness of recommendation motivate our group to build a recommender system for Amazon apps based on our knowledge of personalization techniques and algorithms. We select Amazon dataset because Amazon pays much attention collecting user data by asking them to rating their purchase and provide feedback to their experience. Therefore it's possible to gather plenty of data about each user as well as their purchase history. The more user data collected, the wider range of algorithm can be applied and the output would be more personalized.

1.2. Objective

The project aimed to create a hybrid model so as to offer each user personalized Amazon apps. The hybrid model was designed with 3 personalization approaches: k nearest neighbors with Locality Sensitive Hashing (KNN with LSH), Probabilistic Matrix Factorization (PMF), and Factorization Machine (FM).

1) K Nearest Neighbors were considered while analyzing the sample matrix. Also, the similarity matrix of items was computed by LSH. 2) PMF not only adopted an observation noise and a Gaussian distributed prior associated with each user and item, but also outperformed the standard SVD models. 3) FM predict values from a relation matrix between users and items . Also, by adding extra feature, the accuracy of FM model could be increase. Finally, the project evaluated 3 models with RMSE, accuracy (ACC), and area under ROC curve (AUC).

2. Data

2.1. Overview

The dataset had more than 660,000 records of purchase, each record contained information about apps and users and metadata about them, including description for apps, categories of apps, developer, reviews, star-rating, review time, etc. There were more than 200,000 unique users and 10,000 unique apps involved. In order to create a user-item rating matrix, it was necessary to reduce the dimensionality to avoid memory crash; thus, it was worth to first plot the distribution of user's total purchase as follow:

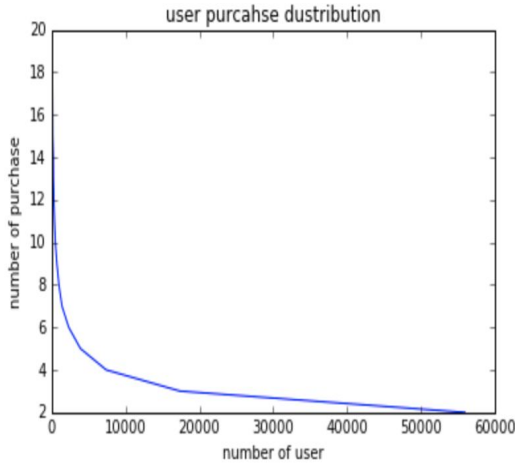

Figure 1.

2.2. Preprocessing

From the figure 1, the data had significant long tail effect. This indicated most users only rated a few items. In fact, more than 100,000 users only rated one item, so it was reasonable to exclude them since there was not enough data to make personalized recommendation. Instead, it was reasonable to treat them as "new user" and recommended them with most popular products, such as a product with highest average rating. Similarly, items with less than 10 reviews were removed from the original dataset because they required more ratings to stabilize their average rates. After these steps, there were only about 5,000 apps and 90,000 users left.

2.3. Clustering by NLP

2.3.1. Cross-Category Recommendation

This step was to cluster similar apps (items) into categories by their descriptions. By grouping similar items, it was possible to compute how every user rated every cluster by averaging their ratings inside that cluster. Thus, it was reasonable to recommend items with top-k average rating in a cluster to a user if the associated predicted rating was high. This procedure was defined as "cross-category recommendation".

2.3.2. In-Category Recommendation

"In-category recommendation" was to predict the rating between each user and item within the same category. For every cluster of items, there were a small subset of users involved on average, so a user-item rating matrix for each cluster was created. These smaller-size matrices could be filled in with predicted ratings efficiently.

2.3.3. Feature Extraction & PCA

By applying NLP techniques-- such as stemming, removing stop words and tokenizing-- the app descriptions looked like the following:

```
words[0]
```

```
'apple selection itunes  app store essentials  selling iphone app years my kids app  by father  under yrs  perfect
fun toddler  toddler buy yet  by ro dub  mama doctor doctor said  no monkeys parents children children monkeys coun
ting game rhyme everybody counting  children math skills color recognition monkeys monkey  objects concepts stop go
 on off  alarm clock rings  lamp  telephone  family portrait touch  reading  words fosters word recognition sentence
 building skills  step learning  singing  music types to  country  rock  pop sing a long  explore  touch  control gam
e options freedom preschooler  application bed  children year pre toddler preschool features touch  peekaboo dog  but
ton stops monkeys children bedroom  shake  monkeys somersaults  press number monkeys watch drop interact objects app
s children  macdonald contact questions  you thanks loeschware llc'
```

Figure 2.

To create a term-document matrix and feature vectors for the following analyzing, Tf-idf-vectorizing and PCA were applied respectively. According to the below singular value plot (figure 3 and 4), it is reasonable to pick the effect low rank k as 100:

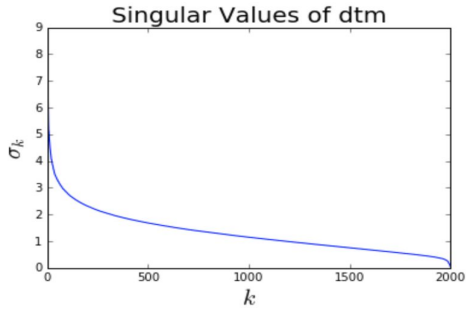

Figure 3.

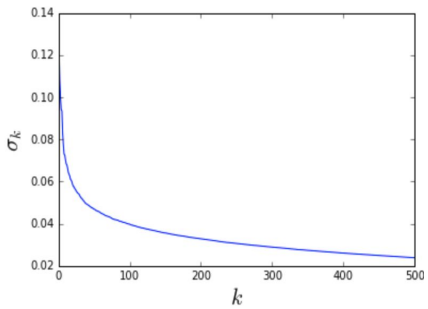

Figure 4.

2.3.4. Algorithm

Among all clustering approaches, k-means algorithm was selected because it was able to generate clusters with meaningful labels efficiently. Elbow method was applied to decide the value of k:

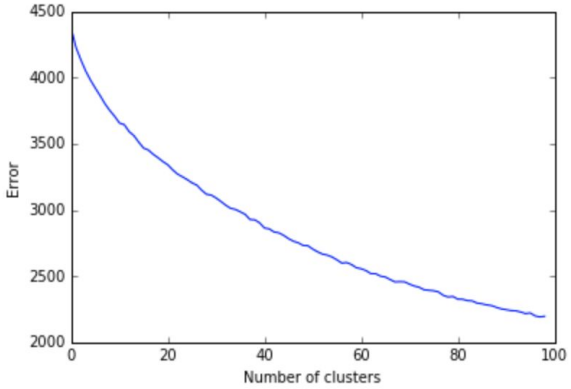

Figure 5.

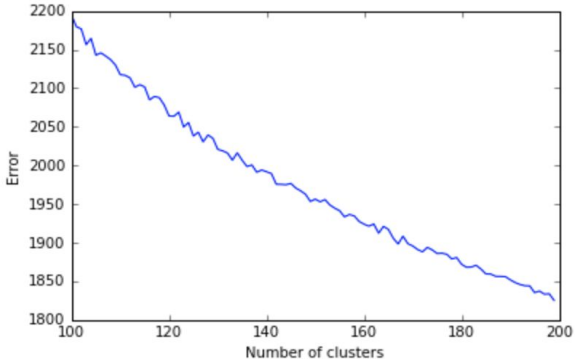

Figure 6.

According to the above plots (figure 5 and 6), the MSE decreased significantly from 4,500 to 2,200 when k increased to 100. However, the MSE only reduced from 2,200 to 1,800 when k increased from 100 to 200 with fluctuation. Thus, apps (items) were clustered into 100 categories referring to value k.

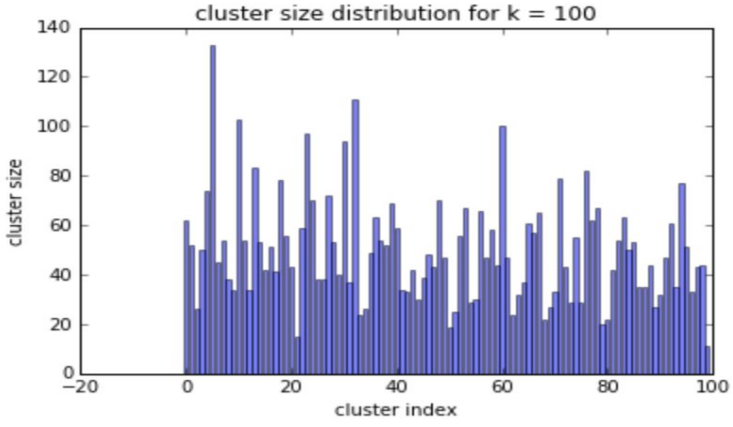

Figure 7.

2.3.5. Result

Notice that the size of each cluster were evenly distributed according to figure 7, and each cluster had some meaningful words to indicate its topic (eg. “score” and “awards” in cluster 0; “adventure” and “games” in cluster 2; “word” and “language” in cluster 4, etc.) according to

figure 8. The largest cluster contained 133 items and 7,579 users who rated at least one item in this cluster. Its size was significant smaller than the original user-item matrix even after removing the unpopular items and inactive users (90,000*5,000). Clearly, breaking down the original matrix and applying personalization algorithms to each sub-matrix would improve the running time and space complexity dramatically.

```
Topic 0:
game play games graphics players gameplay try features modes style help score master goal achievements strategy way h
ours rules awards
Topic 1:
mode modes challenge object ways difficulty picture bonus step play child practice accept hidden timed rounds present
s zen multiplayer experience
Topic 2:
games fish bigfishgames bigfi pc sh mac discover adventure facebook http bigfishtwitter bigfishgamesmobile marketplac
e big www amazon selection type enjoy
Topic 3:
items hints challenge item list timed silhouettes bonus accept modes ways random rounds experience selection picture
 training hiding beginners scenes
Topic 4:
word words search language letters dictionary list game games categories crossword grid scores association sight phra
ses level modes dictionaries vocabulary
Topic 5:
piece jigsaw puzzleboss pieces puzzles facebook support puzzlebossgames captivate photographers photography families
 drawn profiles sizes minute session adults resolution required
```

Figure 8.

3. Models

3.1. k Nearest Neighbors with Locality Sensitive Hashing (KNN with LSH)

3.1.1. Description

LSH was used to improve the efficiency and accuracy of traditional k-nearest neighbor model (KNN). The traditional KNN calculated the similarity of users and items. After calculation of similarity, KNN predicted user-item ratings by the existing ratings that a user given to the k nearest neighbors of specific items. The formula was as follow:

$$\hat{r_{uj}} = \mu_j + \frac{\sum_{i \in P_j(u)} Sim(i,j) * (r_{ui} - \mu_i)}{\sum_{i \in P_j(u)} |Sim(i,j)|}$$

As the size of data grew, the number of computation would increase, since it is required to compute the similarity for every user-item pairs. Therefore, this project introduced LSH approximated method to produce a smaller candidate-neighbor set in advance. This allowed KNN to search neighbors based on the candidate set in a comparably short time. Moreover, LSH was a good choice when encountering sparse data structure (the density of amazon app dataset was 2.67% for cross-category matrix and 2.08% for in-category matrix).

There were four sub-parts for KNN with LSH model:

1) A total number of hash functions *t* was decided (e.g. *t* = 128 in practice). For each item-function pair, a signature represented the minimum hash value among all the hashed users who rated a corresponding item in the past. After doing pairwise calculation for every item and function, a signature matrix composed of minimum hash values was created.

2) The signature matrix was banded into *b* bands where a band contained *r* rows. Hyper parameter threshold decided how to choose a proper value of *b* and *r*. The definition of threshold was the value of similarity *s* at which the probability of being a candidate was 0.5. The formula of threshold was $(\frac{1}{b})^{\frac{1}{r}}$. In this project, the threshold was chosen to be 0.6, since it had a higher accuracy compared to other values. This would be further discussed in the Hyper Parameter Tuning section.

3) If two items shared same signatures in r rows, they would be mapped into one bucket. Finally, all items in the same bucket were considered as a candidate set of neighbors.The candidate set was taken as one input of KNN model. Thus, the model could efficiently calculate the similarity among the candidate set for an item by introducing LSH.

3.1.2. Hyper Parameter Tuning

The training data of KNN with LSH model was divided into four folds for cross validation. In this project, all data of cluster 0 (1,997 users × 60 items) was chosen as the train data. For every unknown user-item rating, average rating of all items are filled in as the baseline. Besides, the baseline reflected an MAE of 1.10 and RMSE of 1.37.

There were two main hyper parameters: number of neighbors for KNN and threshold for LSH. MAE, RMSE, and running time was taken into consideration in our evaluation.

3.1.2.1. Number of Neighbors: k

The following figure 9 and 10 showed the relation between number of neighbors and MAE/ RMSE where threshold of LSH was equal to 0.5.

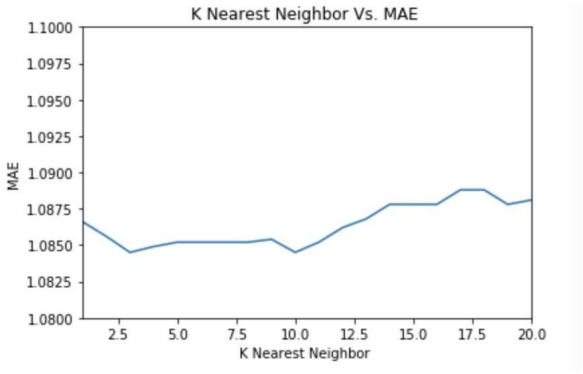

Figure 9.

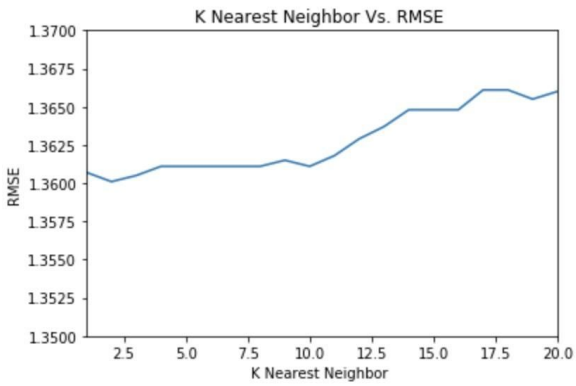

Figure 10.

According to Figure 9 and 10, both MAE and RMSE had lower value when k equalled to 10. Besides, when k was greater than 10, both MAE and RMSE increased significantly. Since when the chosen neighbors were less similar to the target item, the value of error may rise.

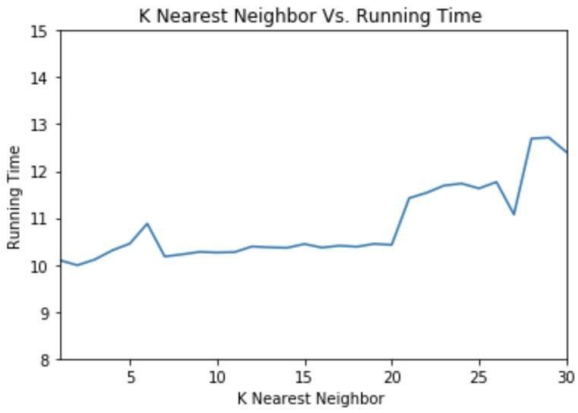

Figure 11.

Moreover, according to the above figure 11, the running time of the process had little fluctuation when k was smaller than 20, but it turned out to increase comparably while k was greater than 20. The reason was that when the number of neighbors became large, more items were thought as the nearest neighbors. This led to more calculations for predicting ratings for the target item.

3.1.2.2. Threshold for LSH

The threshold was defined as the value of similarity *s* at which the probability of being a candidate was 0.5.

The following figure 12 and 13 showed the relation between threshold for LSH and MAE/ RMSE where the number of neighbors for KNN model was equal to 10.

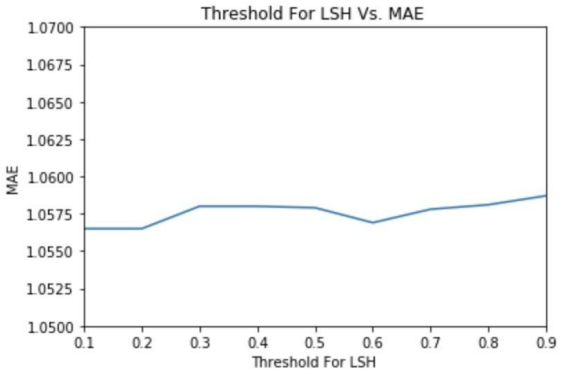

Figure 12.

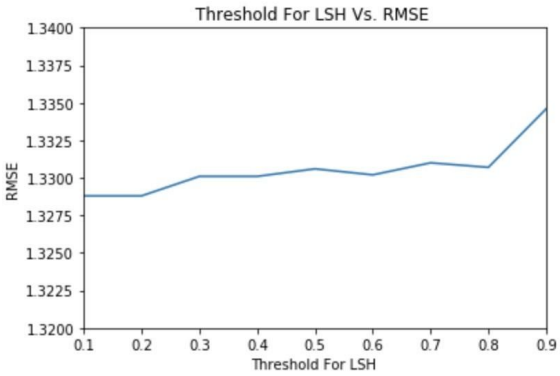

Figure 13.

Referring to figure 12 and 13, when the threshold for LSH increased, the MAE and RMSE would become larger. Since when the threshold increased, the size of candidate-neighbor set would decrease. Moreover, the lower MAE and RMSE occurred when the threshold was around 0.6.

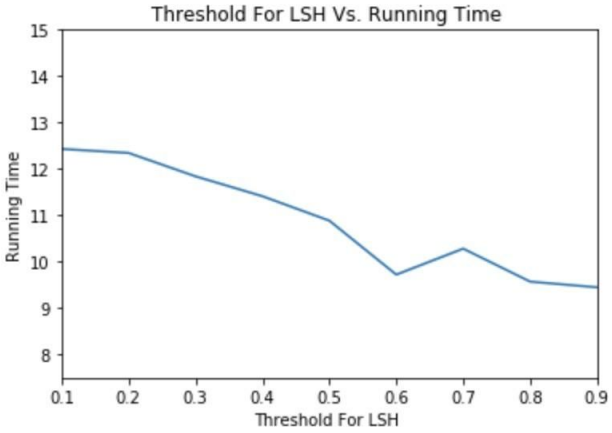

Figure 14.

Besides, according to figure 14, the running time decreased when the threshold for LSH went larger. Because the candidate-neighbor set became smaller while threshold increased, which required less time for computation. As the threshold became smaller and smaller, the running time of algorithm would be as large as the traditional KNN model. In conclusion, it was ideal to make threshold equal to 0.6 in practice.

3.1.3. Evaluation

For model accuracy, when the threshold was chosen to be 0.6 and the number of nearest neighbors was equal to 10, this model reached the lowest average MAE: 1.084 and RMSE: 1.361 (baseline had MAE: 1.10 and RMSE: 1.37). Moreover, the performance with augment data size and coverage of model prediction was evaluated and discussed in the following section.

3.1.3.1. Performance with the augment of data size

There were 100 categories taken as our sample data. In this project, they were classified into several ranges-- [0,20], [20,40], [40,60], and [60,80]-- depending on their size. The following figure 15 showed the relation between running time and category size.

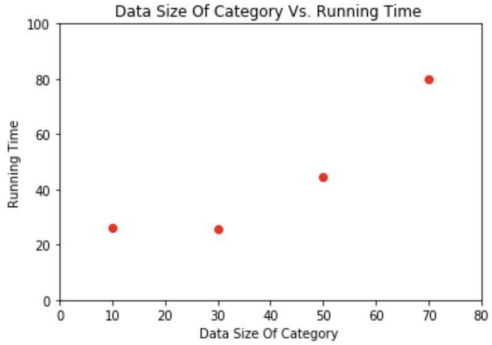

Figure 15.

Referring to figure 15, when the data size increased, running time of the algorithm also increased. Moreover, the running time increased much faster when the data size became greater than 50.

3.1.3.2. Coverage of model prediction

Item-coverage was defined as the proportion of possible recommended items to all users. The coverage was able to offer a precise feedback about whether the recommendation covered relevant items and serendipity, rather than only contained popular items.

$$Item\ Coverage = \frac{number\ of\ possible\ recommended\ items}{number\ of\ all\ items}$$

According to the below figure 16, when the number of recommendation items increased, the coverage would increase, and it grew slower after most of possible items were covered. The model showed a nearly 80% coverage when the number of recommendation items was equal to 5.

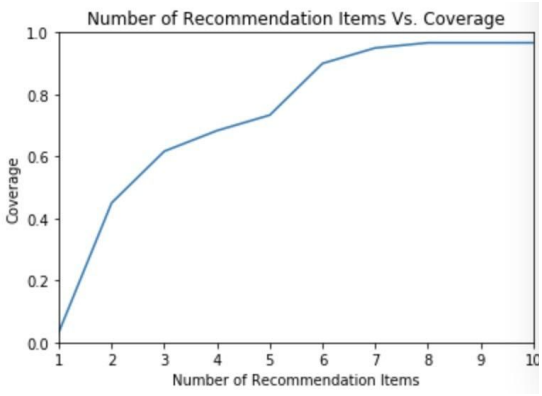

Figure 16.

3.2. Probabilistic Matrix Factorization (PMF)

3.2.1. Description

The PMF-based Matrix factorization technique was well-suited for large and sparse dataset. PMF model not only scaled linearly with the number of observations, but also outperformed the standard SVD models implemented in project part I (Salakhutdinov & Mnih, 2008).

Suppose that there were *M* apps and *N* reviewers; the task was to predict reviewers' preferences matrix *R* where its entry represented r_{ij} (the rating of reviewers i for apps j). In addition, rating r_{ij} could be determined by a linear combination of user and item latent factors, PMF adopted an observation noise and a Gaussian distributed prior associated with each reviewer and app (item).

$$p(R \mid U, V, \sigma^2) = \prod_{i=1}^{N} \prod_{j=1}^{M} [N(R_{ij} | U_i^T V_j, \sigma^2)]^{I_{ij}}$$

This was a joint distribution of user preference matrix *R* and I_{ij} was the indicator function for reviewer *i* rated App *j*.
The reviewer vector *U* and App feature vector *V* had a joint distribution as follows.

$$p(\sigma_U^2) = \prod_{i=1}^{N} N(U_i | 0, \sigma_U^2 I) \qquad p(\sigma_V^2) = \prod_{i=1}^{N} N(U_j | 0, \sigma_V^2 I)$$

The maximization of posterior distribution $p(U, V \mid R, \sigma_U^2, \sigma_V^2)$ could be performed as a minimization task on following regularized squared error with *L2* Norm:

$$L = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{M} I_{ij} (R_{ij} - U_i^T V_j)^2 + \frac{\lambda_U}{2} \sum_{i=1}^{N} \| U_i \|_F^2 + \frac{\lambda_V}{2} \sum_{j=1}^{M} \| V_j \|_F^2$$

where $\lambda_U = \frac{\sigma^2}{\sigma_U^2}$ and $\lambda_V = \frac{\sigma^2}{\sigma_V^2}$ could be computed. Noted that, given all rating that have been observed, when the limit of prior variances went to infinity, the equation was reduced to a SVD objective.

To minimize the objective function above, stochastic gradient descent was implemented. It was reported to be scalable to large dataset, when using a simple gradient descent method with PMF.

3.2.2. Hyper Parameter Tuning

Cross-Validation of five-fold were used in hyper parameter tuning process to receive a better accuracy on all datasets.75% training data was used to test all algorithms. Training data 75%, for

example, means randomly select 75% of the ratings from each category clustering dataset (total of 100 in category and 1 cross category) as the training data to predict the remaining 25% of ratings. The random selection was carried out 5 times independently. Accuracy was measured by rMSE.

To represent the tuning across all data clusters, 3 categories: cat. 0 with (1997 users × 60 items) , cat. 32 (2711 users × 60 items), and cat. 55 (1538 users × 110 items) were chosen as train data sets to visualize the parameter tuning process. Since the performance of the model generally depended on the size of input, which was related to size of category. In PMF model, two hyperparameters, the number of latent factors and regularization term, were tuned.

3.2.2.1. Regularization

Figure 17 showed that rMSE of all three input size PMF models decreased with smaller value of *L2* regularization term lambda. Regularization value 0.001 was chosen by finding the value of hyper parameters that had the lowest rMSE for each category.

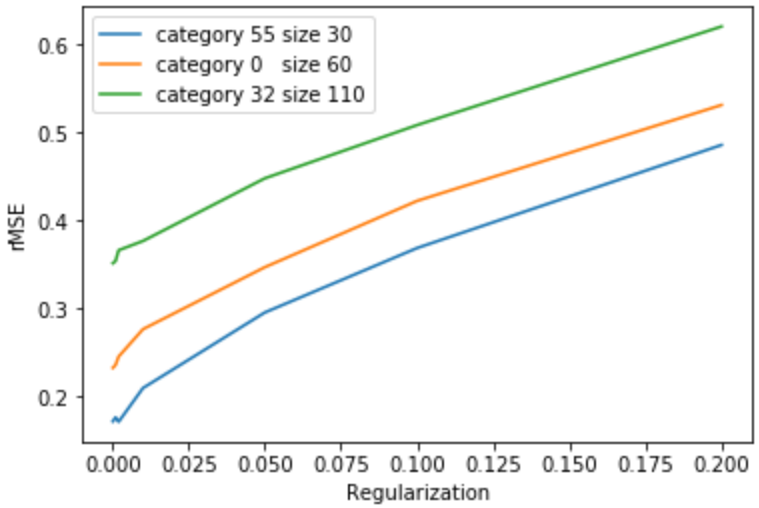

Figure 17.

3.2.2.2. Number of Features

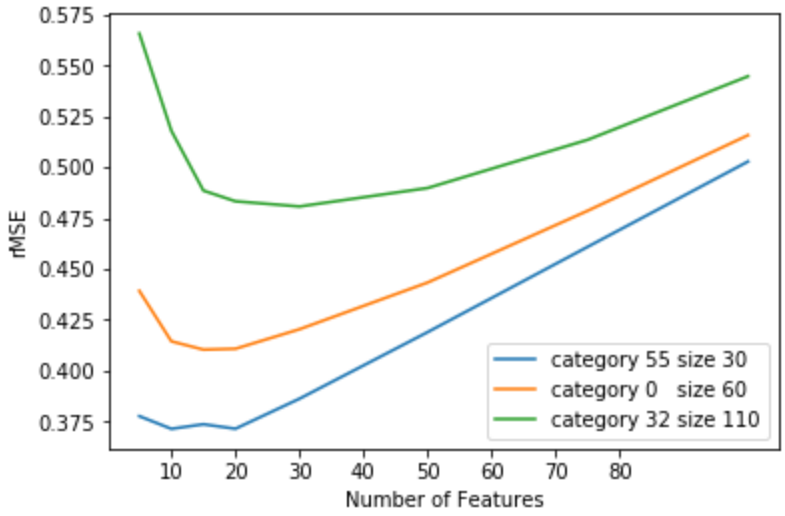

Figure 16.

The number of factors, is a crucial hyperparameter in PMF model. In the process, regularization was set to 0.01. The relationship between number of feature and rMSE was displayed above. As the number of factors (rank of R) increased, the more latent factors for U and V matrices were computed, so accuracy of PMF would first increase and then decrease. For category 55 and category 0, rMSE reached its minimum at feature number is 20, whereas rMSE of category 32 was lowest, when feature number is 30. This result made sense since a larger data set as category 32 need larger rank of R, larger latent factor space of user and iterm matrices. Hence, the number of latent factors was set to 20 for cat. 0, cat. 55 and set to 30 for cat. 32 with regularization term set to 0.001.

3.2.3. Evaluation

3.2.3.1. Scalability analysis with rMSE

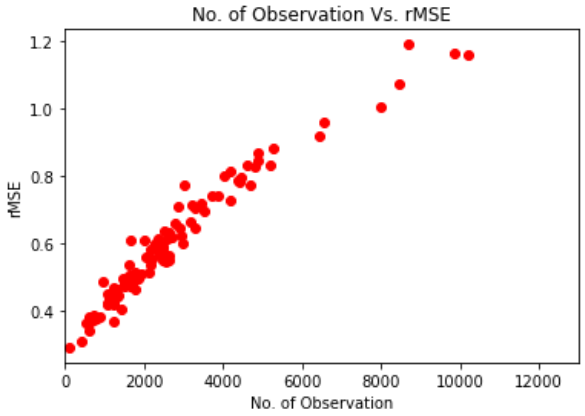

Figure 17.

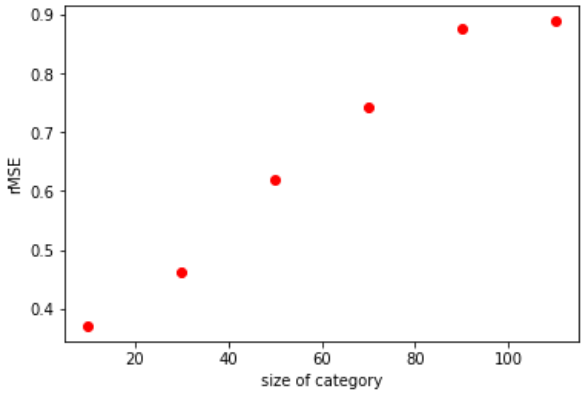

Figure 18.

Number of observation has a strong positive linear relationship with accuracy. When the number of observation in dataset increased, rMSE increased with greater variability. This linear relationship could be observed on size of category with rMSE as well. That was when there were more items included in the user preference matrix *R*, accuracy was generally lower with its variance fluctuating.

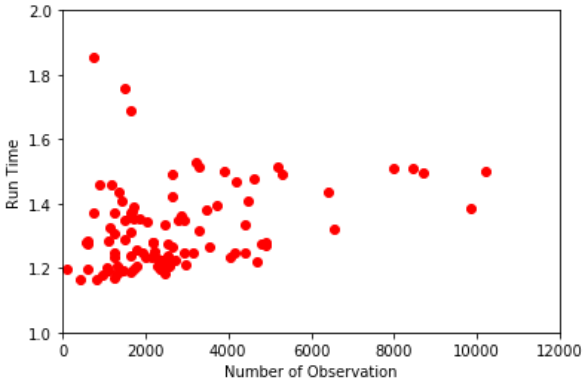

Figure 19.

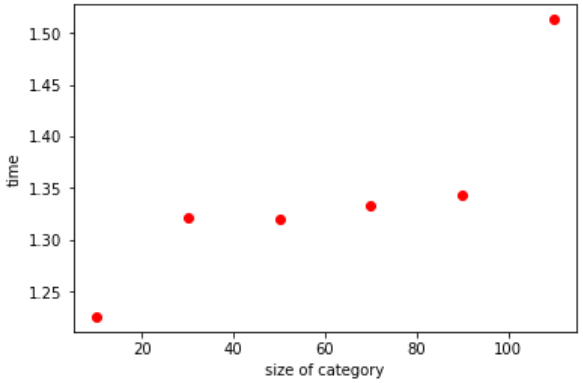

Figure 20.

According to the figure 19 and 20, most of the category data had 1,000 to 4,000 observation ratings, their running time ranged from 1.2s to 1.5s. Time of computation increased with size of input. Maximum computation time on one category is 1.8s with an average time of 1.3s. This running time was relatively fast and consistent across different category data. Notice that for larger number of observations, the run time increased but still kept within 1.6s per data set.

3.2.3.2. Coverage of the model

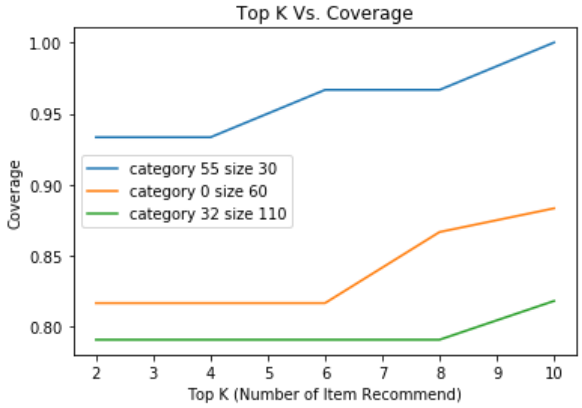

Figure 21.

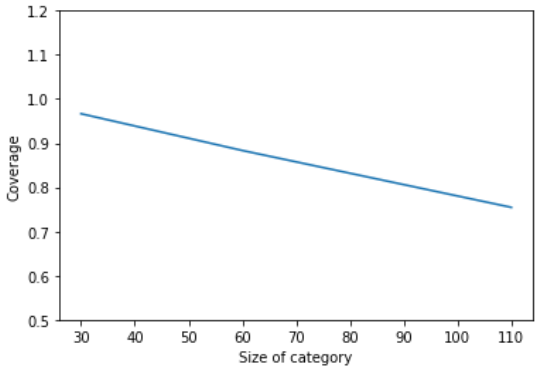

Figure 22.

The above figures shown that as size of input (number of observation and size of category) grew, the coverage percentage dropped. Given trained model of a category, an average 93% of item coverage could be observed. Since the items were classified into 100 categories, there were much fewer items to users. Hence, a prediction of Top 3 items on user rating on one category could successfully cover large portion of items in the category.

3.3. Factorization Machine (FM)

3.3.1. Description

This algorithm aimed to predict rating of amazon apps through FM. The reasons for choosing FM lied in two aspects:

1) The user-item rating matrix was extremely sparse. For example, the density of cross-category matrix was 2.67%, and the average density of in-category matrix was 2.08%.
2) The complexity of FM was linear in terms of the input size and the size of factorization.

Therefore, FM was expected to deal with large and sparse dataset in a fair amount of time. The algorithm was based on Python library, fastFM with ALS regression model. This could be separated into 3 sub-parts: preprocessing, model fitting, and post-processing.

3.3.1.1. Preprocessing

First, every user-item rating pair in the original dataset should be transferred to {"*user*" : …, "*item*" : …, "*rating*" : …} tuples stored in a list object. Second, the list of tuples was converted into a relation matrix. Finally, the relation matrix would be split into train and test (8 : 2) as the following image:

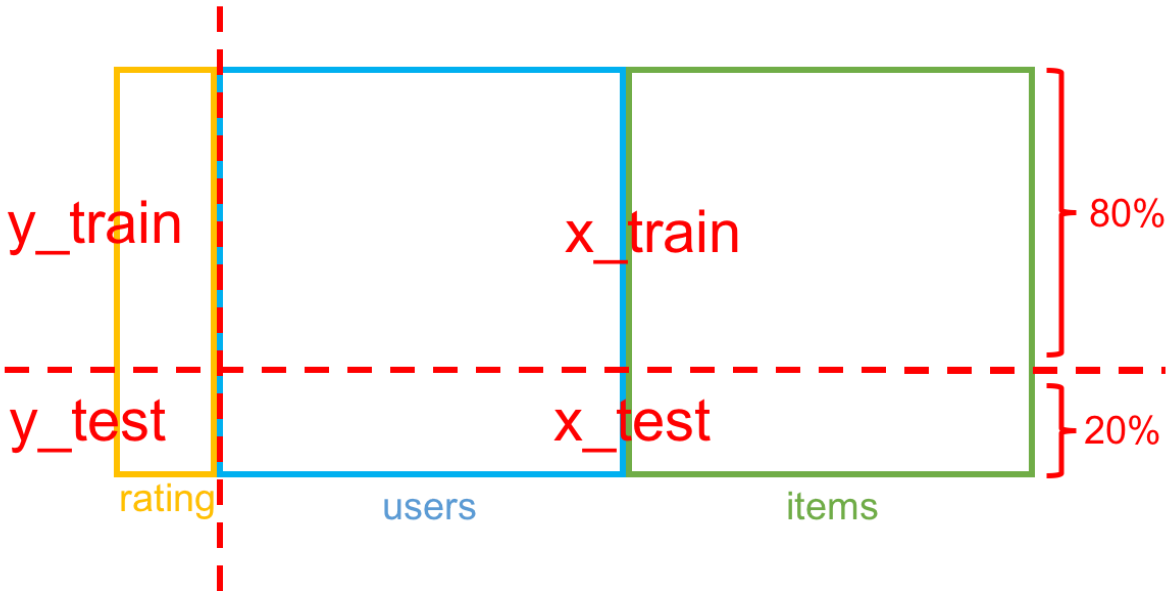

Figure 23.

3.3.1.2. Model Fitting

FM was based on ALS model with hyper parameters: 1000 iterations, 0.1 initial standard deviation, and rank k. First, FM model would fit train dataset from preprocessing part to an ALS regression model. Second, based on the regression model, FM took test dataset as an input and produced the prediction. The algorithm calculated RMSE, ACC (accuracy), and AUC (area under ROC curve) as evaluation. This project defined the rating score that was greater than 3.5 as 1 and 0 for rating scores smaller or equal to 3.5 for ACC and AUC. Hence, the definition of RMSE, ACC, and AUC were:

- RMSE: Define y_i as the true value of test dataset divided by 20% from the original data. Define $\hat{y}_i$ as the predicted rating scores. Let n be the size of test data. Thus, the formula of RMSE was

$$RMSE = \sqrt{\frac{\sum_{i=0}^{n}(y_i - \hat{y}_i)}{n}}$$

- ACC: Since the original ratings were integer numbers, only rating 4 and 5 would be considered as 1. However, for prediction, rating scores larger than 3.5 were treated as 1 since the predicted values were float numbers. ACC was defined as the percentage of correctly predicted values.
- AUC: Since AUC had same condition as ACC, AUC was defined as the area under ROC curve.

Finally, the FM process would return the regression model for later prediction.

3.3.1.3. Post-processing

The goal of this section was to forecast the rating scores for all users to all items. In order to predict all user-item relationships, permutation of all users and items was produced. Moreover,

relation matrix of all users and items was created by similar process in preprocessing part. Lastly, by utilizing the FM regression model from the previous part, the algorithm was able to generate all the relevant ratings.

Based on three parts of the algorithm, in-category prediction processed 100 different item clusters. After the whole prediction, the algorithm concatenated 100 predictions to a user-item rating table. Besides, cross-category prediction works on user-cluster rating pairs. Since there were about 250,000 pairs, they were divided into several parts with size of 30,000. After processing 8 iterations of user-cluster rating tuples, the user-cluster matrix was filled with relevant predicted rating.

3.3.2. Hyper Parameter Tuning

3.3.2.1. In-category Tuning

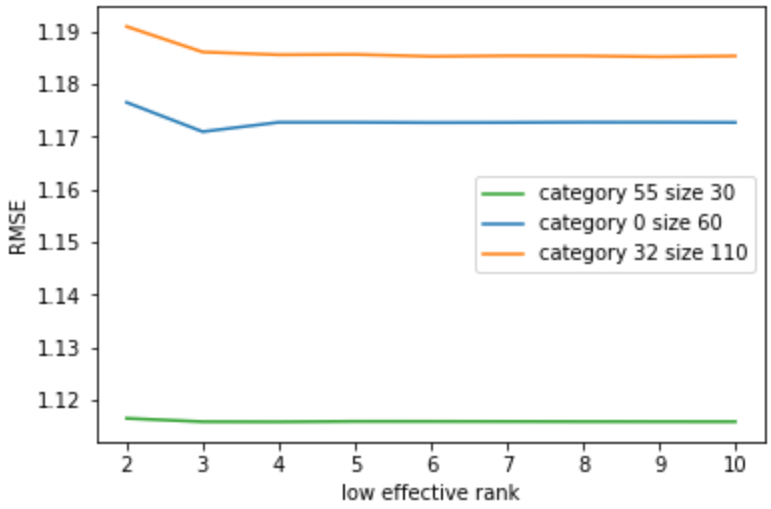

Figure 24.

According to figure 24, the lower the size, the lower the RMSE, because as the size increasing, the sparsity of the data also increases. Obviously, it is relatively harder to predict sparser data. Additionally, the lowest RMSE occurs when the low effective rank is equal to 3. Hence, it is reasonable to choose low effective rank to be 3 as our hyper parameter for in-category prediction.

3.3.2.2. Cross-category Tuning

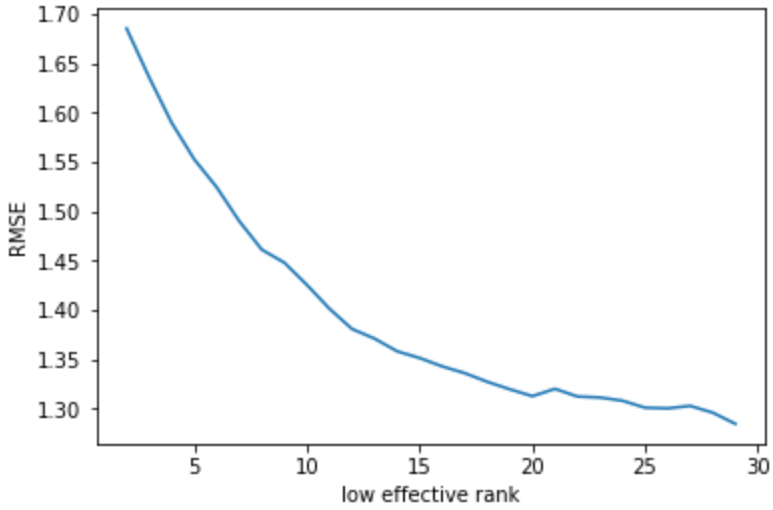

Figure 25.

According to figure 25, the higher the low effective rank, the lower the RMSE. It is obvious that the gradient happened when rank was equal to around 20. Thus, by choosing low effective rank to be 20, the algorithm could generate result in an acceptable time interval with a comparably low RMSE.

3.3.3. Result & Evaluation

3.3.3.1. In-category Result

The following table was the first 5 rows of in-category prediction. Each row was an id of users and each column was an id of items. “NaN” referred that there was no any category having the history that the specific user bought the item. For instance, NaN for user “A00039763E5V43M02Z3YZ” and item “B004A9SDD8” implied that the user did not buy any item that had same category with item “B004A9SDD8”.

2_item	B004A9SDD8	B004ALVL6W	B004ANC00Q	B004ANE2WU	B004AZH4C8	B004AZXRRY	B004BN3YQE	B004CN7Y4G	B004DLNBDA
1_user									
A00039763E5V43M02Z3YZ	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
A00111163LLS4KLYZXNZL	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
A002359833QJM7OQHCXWY	NaN	NaN	NaN	NaN	4.7811	NaN	3.6808	4.2663	NaN
A00236702NOP1FKXVRFN3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
A0026556YRYONJ90H57Z	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 4965 columns

Figure 26.

The histogram of RMSE and number of clusters; and scatter plot of cluster size and RMSE are as follow:

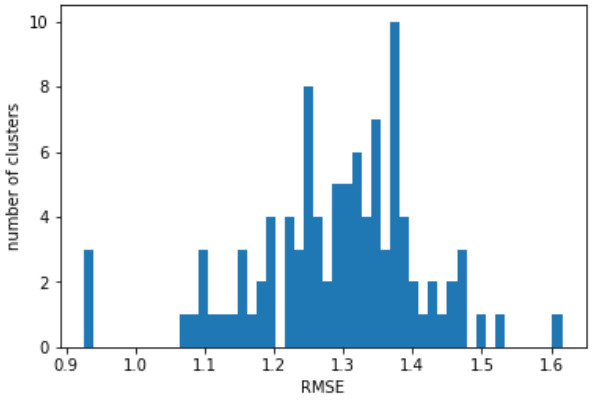

Figure 27

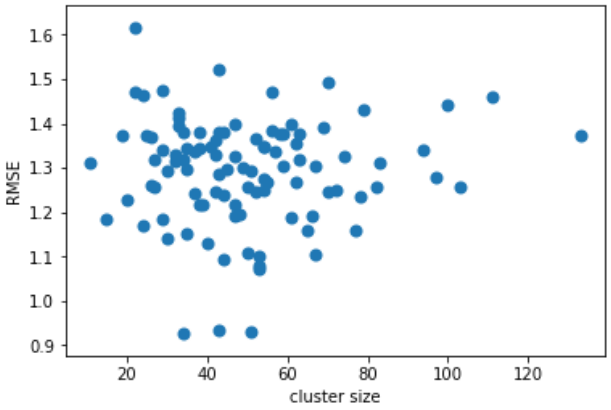

Figure 28

It is obvious to see that there are about 10 clusters having RMSE around 1.38 from Figure 1. Also, most clusters have size 30 to 60 with RMSE 1.2 to 1.4 according to figure 2. The mean of RMSE, ACC, and AUC of this prediction were 1.2914, 72.72%, and 60.29% respectively.

3.3.3.2. Cross-category Result

The following table was the first 5 rows of cross-category prediction. Each row was a cluster (category from 0 to 99) and each column was an id of users. The elements determined a predicted rating from a user to a specific cluster. For example, rating 3.9210 in "cluster 0" row and "A00039763E5V43M02Z3YZ" column refers that users "A00039763E5V43M02Z3YZ" may have an average 3.9210 to every item in cluster 0. Thus, according to this rating, NaNs in previous table could be predicted as 3.9210 if the item belongs to cluster 0.

1_user 2_item	A00039763E5V43M02Z3YZ	A00111163LLS4KLYZXNZL	A002359833QJM7OQHCXWY	A00236702NOP1FKXVRFN3	A0026556YRYONJ90H57Z	A00281722IA1
cluster0	3.9210	5.0000	4.0198	3.9467	4.1705	
cluster1	3.6583	4.6082	2.1124	3.6163	4.7702	
cluster10	4.0483	5.0000	3.9805	4.0668	4.3816	
cluster11	3.8421	3.6377	5.0000	3.8893	2.9910	
cluster12	3.8788	3.9154	3.4663	3.8579	4.1632	

5 rows × 91533 columns

Figure 29.

The mean of RMSE, ACC, and AUC of this prediction were 1.6765, 68.36%, and 57.75% respectively.

3.3.3.3. Evaluation

From the ACC and AUC scores of in-category and cross-category prediction, FM was an accurate model according to the high accuracy and low MSE. The model was allowed to deal with huge matrix. Although the huge size would easily cause memory error, by splitting data into small parts (categories), FM could still work successfully in a personal laptop with high ACC, AUC, and low RMSE.

From figure 27, most clusters locate in the center with RMSE within 1.2 to 1.4, and only a few datasets are outliers, which indicates the clustering algorithm works successfully. Besides, according to figure 27 and 28, the cluster with largest RMSE around 1.6 has a smaller size. Theoretically, smaller sample size would lead to larger variation.

In addition, since in-category prediction processed much similar items and the input contained less missing user item pairs, it would have a lower RMSE (1.2914). On the other hand, there was less similarity among each category of items; therefore, RMSE for cross-category prediction would be higher (1.6765).

Moreover, although the FM could deal with different features such as timestamp, our original data has too many missing time stamps. It turns out that adding timestamps would not decrease the RMSE significantly, but lower down the running time. Therefore, the algorithm only took user item and rating into consideration.

Lastly, since FM had a great accuracy score, FM model would have a larger weight when working on a hybrid model recommendation.

3.4. Hybrid Model

3.4.1. Description

The mixed offline recommender system is consisted of two levels recommendation: in-category and cross-category.

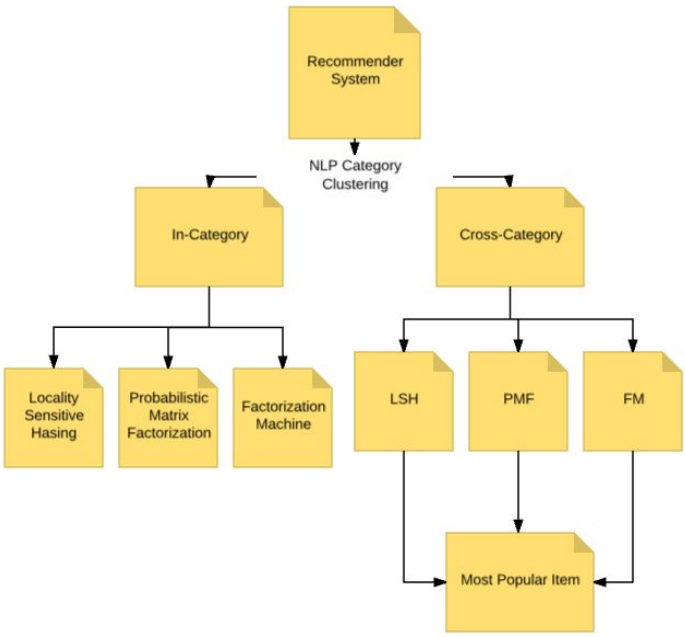

Figure 30.

3.4.1.1. In-category Recommendation

For in-category recommendation, three algorithms were implemented: KNN with LSH, PMF, and FM in parallel. Three models are trained independently for each category and store the prediction results in local files. To combine the results of algorithms, a weighted sum over these predictions of different models was calculated.

$\hat{r}_{uj} = \sum_{i=1}^{q} \alpha_i \hat{r}_{uj}^{i}$ where r is the individual entry in user-item matrix, and α_i is calculated by the ratio of the models accuracy. This weighted sum method reduced the overall variance of these models (Vintch, 2017). For a reviewer [A1LTYRWPRBZ1QV] who has purchased and reviewed a smashing Jack-O-Lanterns App in category 0, the three models gave user predicted ratings of all items in this category. Those three model predictions were then combined by the weighted sum formula, and return the top-k App in this category 0 to recommend. Some of the item would be Sudoku Puzzle, which had an overall prediction score 4.13; 3D Golf Game, score 4.27; and Star Wars: Jedi Knight, score 4.0. It was easy to see the reasoning behind those recommendations. Those recommendations were all games that shared some features of Jack-O-Lanterns.

3.4.1.2. Cross-category Recommendation

As for cross category system, Apps were recommended to user by a sequential hybridization model. First, NLP clustering method was implemented as feature augmentation. It created the first feature set which was user average rating of category so that the method provided input to LSH, PMF and FM. The project predicted the preference category of a user, by measuring similarity between those clustering categories, which the user has purchased item and categories that a user has yet explored. At the last step, the project calculated the most popular items in the preference category and return them as a set of cross-category recommendation.

3.4.1.3. Combination System of In-category and Cross-category

The mixed recommendation result from cross category NLP clustering and in category with parallel models are presented to a user simultaneously. Due to the nature of categorization, the density of user-item matrix was increased significantly. The system also introduced diversity in recommending items and maximizing multiple objectives of the recommendation.

The advantage of this mixed system was obvious. For example, a recommended APP list based only on content-based models might find tracks that described similar to things reviewers like, but would not be very diverse or surprising; whereas, a pure memory-based KNN model might be far off a reviewer preferred category. Combine both to a list base on category can effectively eliminate the large uncertainty of guessing in KNN as well as introducing novelty of the content. Besides, producing novel recommendation, the system also solved the cold start problem. For user with a few item reviews, the system provided similar items and most popular items from similar category to predict user's preference. For new user, since there was no record on the offline system, it would randomly provide from precomputed result of most popular item in the dataset.

3.4.2. Evaluation

A subset of Amazon data was randomly selected to show the impact of hybridization on MSE. In order to predict star ratings, predicted values of three models were combined in two ways. First, the ratings were combined by averaging predicted values from 3 models. Then ratings were computed by a weighted average result of three models, where the weight is inverse proportional to the individual MSE of each model. That was for a given model with large MSE, weight alpha would be small.

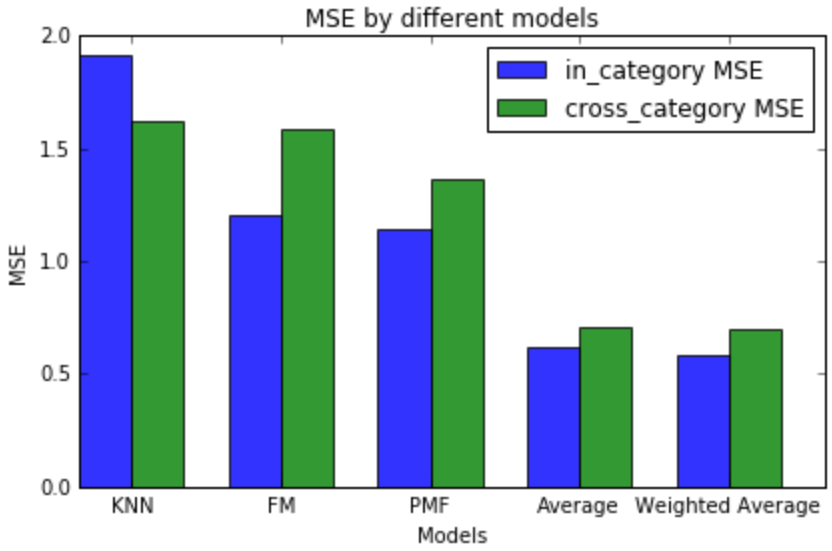

Figure 31.

The MSE values of three algorithms, average hybrid, and weighted average models were shown in figure 31. The two hybrid models had relatively small accuracy variation between them, whereas they achieved much lower MSE over the dataset than KNN, FM, and PMF alone. It was observed that the parallel hybridization impacted the recommendation results significantly. This observation demonstrated that fusing three models together improved the recommendation accuracy.

4. Conclusion

This project aimed to develop a recommendation system for Amazon apps based on a hybrid model. The recommendation based on purchase history was divided into two parts: in-category and cross-category recommendation. For each part, the hybrid model was implemented to offer a precise prediction of unknown ratings for user-item pairs, and then recommend top-k items that the target user might be most interested in. Specifically, hybrid model was composed of three advanced algorithms: K Nearest Neighbor model based on Locality Sensitive Hashing (KNN

with LSH), Probabilistic Matrix Factorization (PMF), and Factorization Machine (FM). The weight of each algorithm was based on the percentage of the accuracy and coverage performance. Thus, the mean of weighted prediction ratings became more convincing.

Three advanced algorithm were highly accurate and efficient compared with the traditional methods, such as KNN and SVD used in the past. Even though each algorithm contained constraints while fitting into datasets, the hybrid model could evaluate the algorithms based on the individual weights. The weights guaranteed the prediction would not be influenced by possible constraints.

Moreover, some potential improvements were taken into further consideration. First, most procedures were computed offline; thereby, when models were wished to process under online situations, efficiency should be necessary improved without decreasing the accuracy. Second, before providing users with recommendation, all predicted ratings should be calculated by three algorithms and be stored in advance. This would occupy a large amount of memory spaces. Therefore, a more appropriate way was to utilize parallel or pipeline work flows in order to achieve a balance between memory cost and running time cost. In conclusion, despite the fact that there were some drawbacks for this model, this recommender system could be optimized furtherly. Also, the system was able to offer highly relevant recommendations, but also provide users with serendipity items.

Reference

R. Salakhutdinov and A. Mnih (2008). Probabilistic Matrix Factorization. *Advances in Neural Information Processing Systems, 20* (07), pp. 1257-1264.

Vintch, B. (2017). Personalization Theory & Application Lecture 7 [PowerPoint slides]. Retrieved from https://piazzaresources.s3.amazonaws.com/j6z51k3cycr6af/j9634pnkqdv73f/7_content_based.pdf?AWSAccessKeyId=AKIAIEDNRLJ4AZKBW6HA&Expires=1513199211&Signature=dbQMyi5Pu74ESZJ9FujGzJnVJvQ%3D.