Machine Learning

<u>Project Report</u>

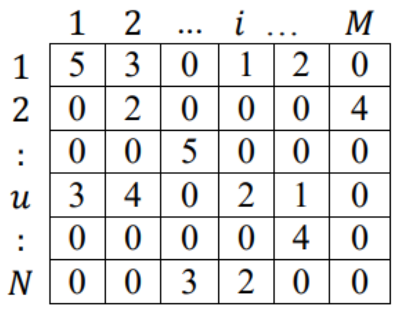
Recommendation System

Student Name	Student ID	Student Email
Nguyễn Tuấn Long	20214963	long.nt214963@sis.hust.edu.vn
Nguyễn Hoàng Anh	20214946	anh.nh214946@sis.hust.edu.vn
Trần Hoàng Anh	20210023	anh.th210023@sis.hust.edu.vn
Bùi Hải Dương	20214953	duong.bh214953@sis.hust.edu.vn

Table of Contents

1. Presentation of the project

- A Recommendation System (Recommender System) is a system designed to provide suggestions for items that are relevant to a specific user.
- The recommendation problem can be transform to a mathematical form as below:
    - Given a finite set **U** containing **N** users $U = \{u_1, u_2, ..., u_N\}$ and a finite set I containing **M** items $I = \{i_1, i_2, ..., i_M\}$.
    - The relation between U and I can be described by a ratings matrix (user-item matrix) $R = [r_{ij}];\ i = 1...N,\ j = 1...M$. Where the value of r_{ij} is the ratings of user i to the item j.If the user i hasn't interacted with the item j, r_{ij} = 0. Below is an example of what R looks like.



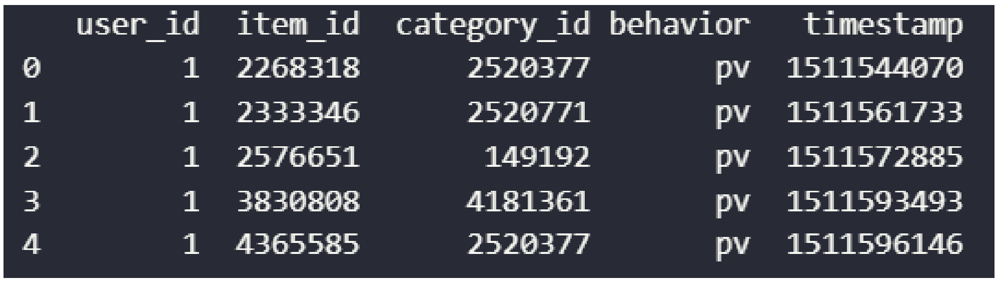
$$\begin{array}{c|cccccc|}
 & 1 & 2 & ... & i & ... & M \\
\hline
1 & 5 & 3 & 0 & 1 & 2 & 0 \\
2 & 0 & 2 & 0 & 0 & 0 & 4 \\
: & 0 & 0 & 5 & 0 & 0 & 0 \\
u & 3 & 4 & 0 & 2 & 1 & 0 \\
: & 0 & 0 & 0 & 0 & 4 & 0 \\
N & 0 & 0 & 3 & 2 & 0 & 0 \\
\hline
\end{array}$$

    - The recommendation temp to predict all the unknown ratings of an user u_a then decide K new items having highest ratings to suggest to u_a

2. Selection of dataset.

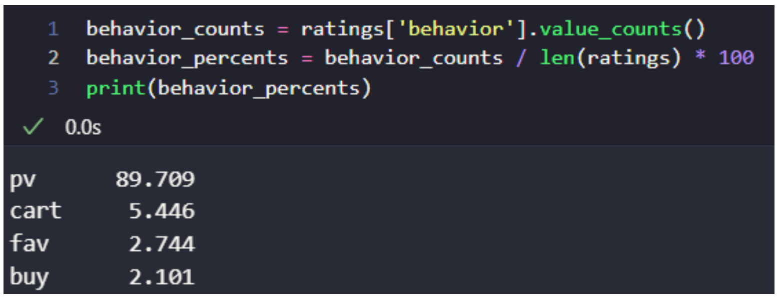
- The dataset we used is the first 100 thousand lines of "UserBehavior.csv" published by Taobao on the website : https://tianchi.aliyun.com/dataset/649.
- The dataset consists of 5 attributes: user_id, item_id, category_id, behavior, timestamp. The column 'behavior' can have 4 values {'pv', 'fav', 'cart', 'buy'}

corresponding to 4 actions: visit, favor, add to cart and buy. Below is a small part of the dataset:



```
   user_id  item_id  category_id behavior   timestamp
0        1  2268318      2520377       pv  1511544070
1        1  2333346      2520771       pv  1511561733
2        1  2576651       149192       pv  1511572885
3        1  3830808      4181361       pv  1511593493
4        1  4365585      2520377       pv  1511596146
```

- We compute the percentage of each behavior to encode each behavior to a reasonable score. Below is the percentage of each behavior:

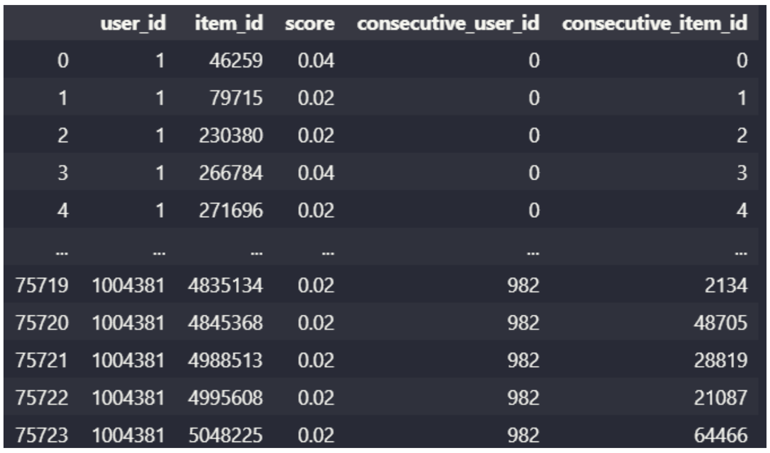


```
behavior_counts = ratings['behavior'].value_counts()
behavior_percents = behavior_counts / len(ratings) * 100
print(behavior_percents)
```

```
✓ 0.0s
```

```
pv      89.709
cart     5.446
fav      2.744
buy      2.101
```

- Because we want to increase the behavior *'buy'* of users, we weighted it by the highest score. Here is the weights for each behavior $score = \{'pv': 0.02, 'fav': 0.5, 'cart': 0.3, 'buy': 0.9\}$
- Moreover, since *user_id* and *item_id* are not in consecutive order, we created a mapping from original IDs to new consecutive IDs.



	user_id	item_id	score	consecutive_user_id	consecutive_item_id
0	1	46259	0.04	0	0
1	1	79715	0.02	0	1
2	1	230380	0.02	0	2
3	1	266784	0.04	0	3
4	1	271696	0.02	0	4
...	...	...	...	...	...
75719	1004381	4835134	0.02	982	2134
75720	1004381	4845368	0.02	982	48705
75721	1004381	4988513	0.02	982	28819
75722	1004381	4995608	0.02	982	21087
75723	1004381	5048225	0.02	982	64466

- Finally, we transformed the dataframe to an *user-item matrix* **R** as mentioned above with the row is *consecutive_user_id* and column is *consecutive_item_id*, r_{ij} is calculated by taking the sum of all scores which the *user i* gave to *item j.*

- The matrix **R** ‘s dimension is *983x64467* indicates that there are 983 unique users and 64467 unique items. However, **R** is very sparse since the known values only take ~ 12%.
- After that, we normalize **R** by filling unknown scores by its row’s mean value then subtract all elements in the matrix by their row’s mean value.
- From here, we will work with this *user-item matrix* **R.**

3. Proposal Algorithms.

We propose 3 algorithms to solve this problem:

- Collaborative filtering user-based
- Collaborative filtering item-based
- Collaborative filtering model-based with matrix factorization

In our case of dataset, Matrix factorization seems to be the best algorithm because it can handle sparsity very well. On the other hand, theoretically, Collaborative filtering user-based and item-based are not very effective due to the sparsity of our data.

4. Implementing the algorithms used for solving the problem.

4.1. Collaborative filtering user-based.

- In this algorithm, we assumed that an user may also want items which many others similar to him like. So there are 2 main steps in this algorithm: Determine k-nearest neighbors (KNNs) for each user, and after that calculate predicted scores for all unknown scores for each user.

- First, recall that we had an user-item matrix **R**. To find the similarity between 2 users, we used a value called cosine similarity. Cosine similarity *sim* between 2 users i and j can be calculated by: $sim_{ij} = \frac{<U_i, Uj>}{||U_i||\,||U_j||}$ $where\ u_j,\ u_j$ are row vectors corresponding to user *i* and user *j*. Then the k-nearest neighbors for each user can be obtained by sorting the value of cosine *sim* in descending order.
- Second, after we get all the k-nearest neighbors for each user, we calculated the predicted scores user i give to item j by

$$r_{ij} = \frac{1}{h}\sum_{n\in K} r_{nj} \times sim_{ni};\ h = \sum_{n\in K} sim_{ni}$$

- If in worst case scenario, no neighbor of KNNs of user i has rated j, $r_{ij} = 0$.
- So, the main concern in this algorithm is how to choose *k*. We started by choosing a popular value *k* = 5, then we created a graph showing a function of *k* and *RMSE* to find the optimal value of *k* (more details about how to calculate RMSE and result graph are written below in part 5).

4.2. Collaborative filtering item-based.

The idea in this algorithm is similar to the approach of Collaborative filtering UserBased. The idea is very simple: an user may temp to like similar items to what he liked in the past. So, instead of finding KNNs for each user, we find KNN for each item, then use known scores of these neighbors to fill in unknown values for each item.

To compute the similarity between 2 item *i* and *j*, we use the same formula for cosine similarity sim:

$sim_{ij} = \frac{<I_i, I_j>}{||I_i||\,||I_j||}$ $where\ I_j,\ I_j$ are column vectors corresponding to item *i* and item *j*.

Then, we get all the k-nearest neighbors for each user, we calculated the predicted scores user *i* give to item *j* by

$$r_{ij} = \frac{1}{h}\sum_{m\in K} r_{im} \times sim_{mj};\ h = \sum_{m\in K} sim_{mj}$$

The main problem here is the same as in the Collaborative filtering UserBased method: we need to find an optimal value for *k* KNNs. So we also draw a graph to visualize a function of k and RMSE (more details about how to calculate RMSE and result graph are written below in part 5).

4.3. Collaborative filtering model-based with matrix factorization (MF).

The idea of MF is that we will find 2 embedding vectors **U** and **V** of **R** such that **R ~ U x V** (x is a matrix multiplication operation).Then, unknown scores that user *i* give to item *j* can be calculated based on the result **U x V**.

$Given\ R \subset M_{nxm},\ find\ U \subset M_{nxk},\ V \subset M_{kxm}\ such\ that$:

$R \sim U \times V + \lambda_1 r(U,V) + \lambda_2 g(U,V)$

$where$: $r(U,V) = \frac{1}{N}\sum_i ||u_i||^2 + \frac{1}{M}\sum_j ||v_j||^2$ is called regularization term

$g(U,V) = \frac{1}{MN}\sum_{i\in N}\sum_{j\in M} < u_i\ , v_j >^2$ is called gravity.

$\lambda_1, \lambda_2\ are\ 2\ regularization\ coefficients.$

Then the total loss is then obtained by

$$\frac{1}{|\Omega|}\sum_{(i,j)\in\Omega}(r_{ij} - < u_i, v_j >)^2 + \lambda_1 r(U,V) + \lambda_2 g(U,V)$$

In practice, we created 2 model: one without the bias term $\lambda_1 r(U,V) + \lambda_2 g(U,V)$ in its loss, one with the bias term. The purpose is to find out how the bias term affects the model's performance.

In our first model (without bias term), we first randomly initialized **U** and **V** with a fixed random dimension *k* and standard deviation between its values *std*. After that, we tried to minimize the loss $\frac{1}{|\Omega|}\sum_{(i,j)\in\Omega}(r_{ij} - < u_i, v_j >)^2$ above by AdamOptimizer. Then we tested the model with several dimensions *k* and standard deviation *std* in order to find the best value for them.

For the second model (with bias term), we did the same as the first model: minimize the loss $\frac{1}{|\Omega|}\sum_{(i,j)\in\Omega}(r_{ij} - < u_i, v_j >)^2 + \lambda_1 r(U,V) + \lambda_2 g(U,V)$ by using AdamOptimizer. However, this model has much more hyperparameters likes: k, std, λ_1, λ_2, so we must run more times to choose the best value for them.

5. Evaluation.

5.1. Method.

- In order to evaluate, we write a function **split_matrix**. This function splits matrix **R** by a chosen ratio *rate* into 2 matrices that have the same size: **train** and **test**.

- To be more clear, for each column of R, we find the number of non-zero elements then randomly select *rate* of these non-zero elements to move to the test set. The train set will have the remaining values.
- We use the same train and test set for all algorithms to compare the loss more accurately.
- Another point worth mentioning is the measurement of predicted values accuracy. We used **Root Mean Square Errors (RMSE)**. In detail, RMSE is only calculated based on **known values** (non-zero values) in the actual set and their corresponding values in the predicted set. The smaller RMSE is, the more accurate the model is.

a. Collaborative filtering UserBased

- After splitting the train and test set, we use *split_matrix* function (mentioned above) to split the train set into 2 sets: train and valid. We fed the model by the train set
- RMSE of the train set was calculated based on the differences between the actual score in the valid set and the corresponding predicted value by the model.
- RMSE of the test set was obtained from the predicted values by the model and the actual value in the test set. Note the model needs to predict values in the test set based only on the train set.

b. Collaborative filtering ItemBased

- The evaluation has been done similar to what we did in Collaborative filtering UserBased. However, since the difference between number of users and items is very large (983 unique users compared to 64467 unique items), the algorithm runs very slow and exceeds the time limit (15 mins) we set. The reason is that the model must calculate similarity between an item vector and 64466 other vectors.

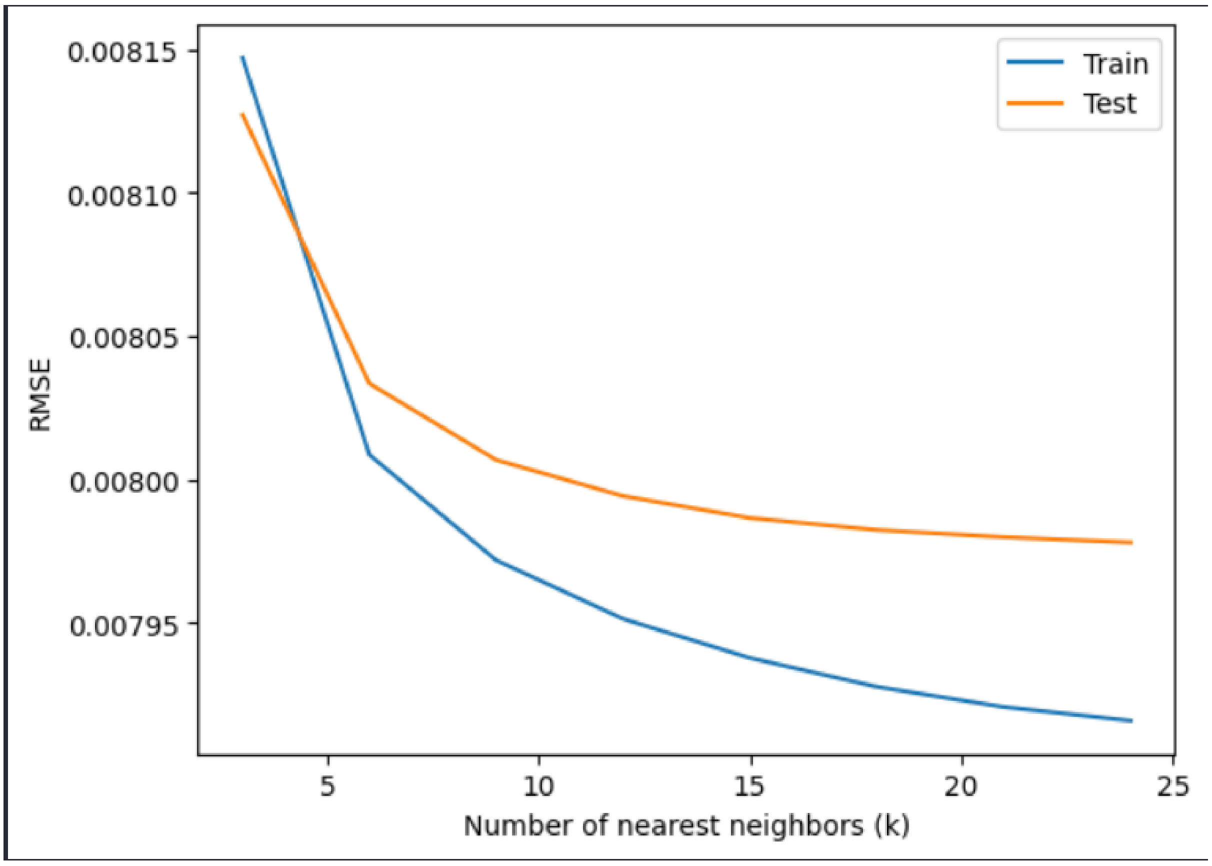
c. Matrix Factorization

- To measure the model's performance, first we used the train set to determine **U** and **V** such that the loss is minimal by AdamOptimer.
- After that we calculated RMSE between the train/test set and the product **U** x **V** and drew RMSE based on the number of iterations.
- One important thing is that we only calculated RMSE on elements that are non-zero in the actual matrix and those corresponding elements in the predicted matrix.
- Finally, we tried several different hyperparameters to obtain the optimal values for each (Detail result can be found below).

5.2. Optimal Hyperparameters

a. Collaborative filtering UserBased

- After we have observed the graph of the *RMSE* train and test depend on *k* KNNs, we can clearly obtained that the optimal range for *k* is [8;15].



b. Collaborative filtering ItemBased

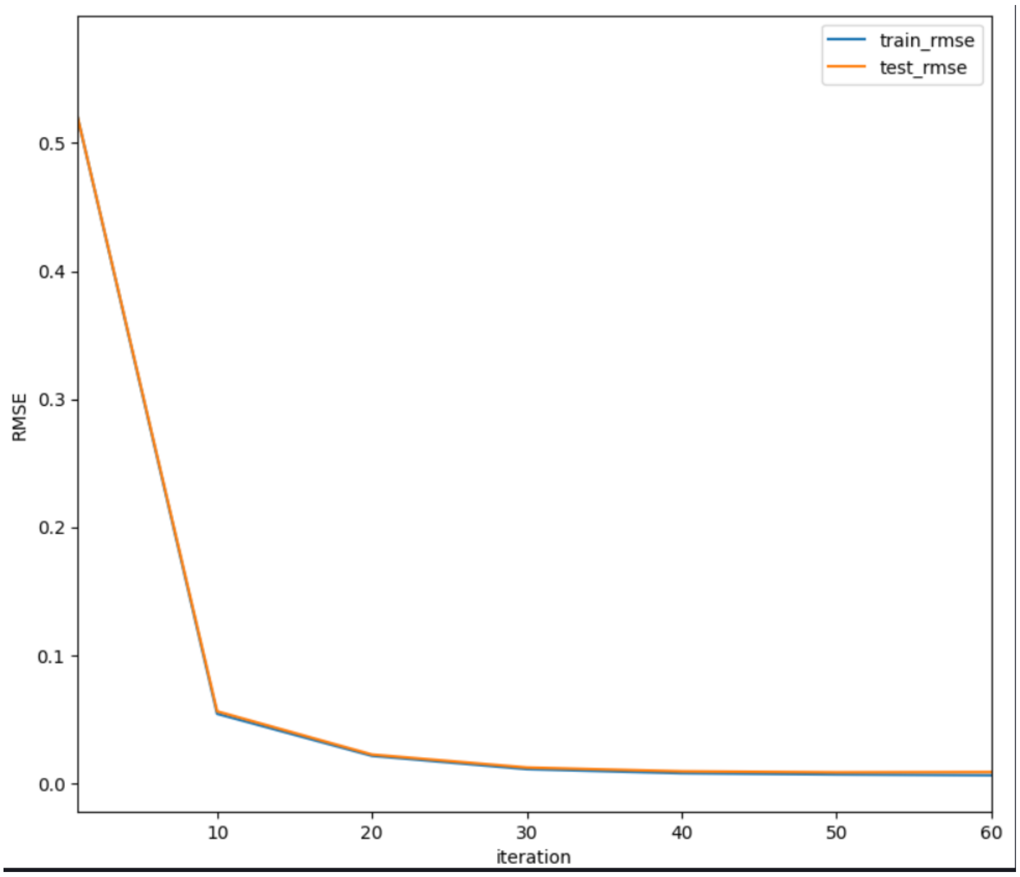
- Because this method has exceeded our time limit (15 mins), we cannot conclude anything about this algorithm.

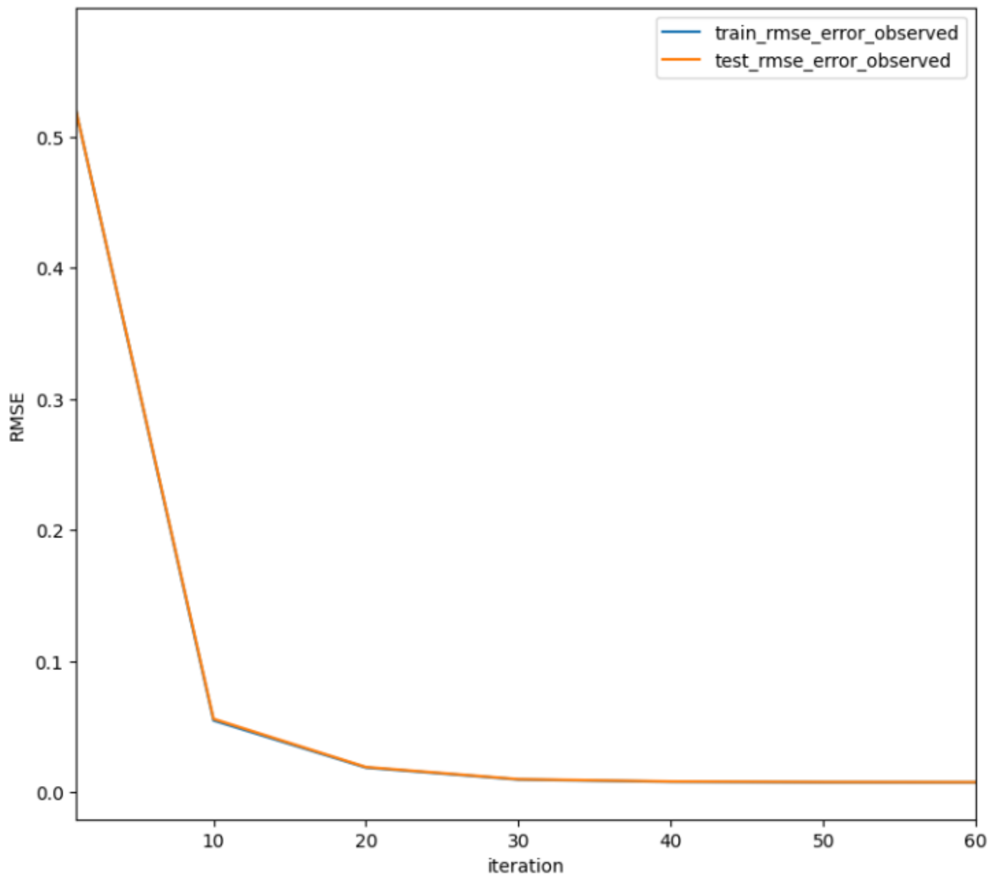
c. Matrix Factorization

- First, we fixed the number of *iteration* = 200, *std* = 0.3 then changed *dim* - embeddings dimension. To save time, we only find optimal dim for the model that doesn’t have bias term and took that value as the optimal value of *dim* for the bias_term_model

dim	20	25	30	35	40	45
Best test RMSE	0.011	0.010	0.010	0.009	0.008	0.009

- Second, we fixed *dim* = 40 and *iteration* = 60 (Because in the graph, iteration more than 60 didn't decrease test RMSE significantly. In some cases, it even increased the test loss) then changed the value of *std*. Finally we obtained the optimal value for *std* is 0.3.
- Then, we switched to the *bias_term_model* and did similar to regularization parameters: change a parameter to be optimal and fixed the others until no significant improvement in test RMSE is observed.
- After setting all the hyperparameters to be optimal, the result, unsurprisingly, is that the bias_term_model has a smaller RMSE test than the normal model 's.
- However, both models have the test RMSE very similar to its train RMSE.
- Below are 2 graphs representing *normal model* and *bias_term model* respectively.


train_rmse
test_rmse
RMSE
iteration
0.0
0.1
0.2
0.3
0.4
0.5
10
20
30
40
50
60


train_rmse_error_observed
test_rmse_error_observed
RMSE
iteration
0.0
0.1
0.2
0.3
0.4
0.5
10
20
30
40
50
60

5.3. Comparing the algorithms

- After setting all hyperparameters to be optimal in every algorithm, we compare *running time(s), test RMSE and train RMSE*.
- The comparison was based on 10 runtimes, each time a pair of train and test set was generated randomly (but all the algorithms use the same pair for each runtime).

Algorithm	Min Time	Max Time	Min Test RMSE	Max Test RMSE	Min Train RMSE	Max Train RMSE
User Based CF	248.5	295.6	0.008	0.011	0.007	0.011
MF	217.6	253.7	0.007	0.009	0.003	0.006

5.4. Explaining the results

The result is a surprise for us. Because theoretically , in our case, the dataset is very sparse, MF has been proven to be more accurate and effective. However, UserBasedCF still can provide quite good results and was not overperformed by MF.

In some cases, although the train RMSE was so small, MF still had a not very small test RMSE. The reason is that maybe in these cases, overfitting has appeared.

However, MF still had average runtimes smaller than UserBased CF’s.

6. Future works

- Due to our limitation in machine learning, the algorithm we found is still very simple and is not very practical. We had some difficulties in the evaluation process in which we did not fully understand how to measure the model ‘s performance on the train set.

- However, we also found a very promising GraphBased approach which is a meta-path in Knowledge Graph for this problem. In the future, we will do more research in this field.
- Finally, we would like to express our deepest gratitude to our instructor Khoat Than Quang for guiding us this semester.

7. List of tasks

Researching UserBasedCF	Nguyễn Hoàng Anh, Trần Hoàng Anh, Nguyễn Tuấn Long
Researching ItemBasedCF	Nguyễn Hoàng Anh, Bùi Hải Dương
Researching MF	Nguyễn Hoàng Anh
Data Preprocessing	Nguyễn Hoàng Anh
Writing Report	Nguyễn Hoàng Anh, Nguyễn Tuấn Long
Writing README.md	Trần Hoàng Anh