BIA 667 Final Project Report

Group 4 - Qihao Zhang, Yuan Yang

1. Problem Description

Since the COVID-19 outbreak in early 2020, people have to keep social distance and stay at home. At the same time, shopping online is the best way to purchase goods while staying safe. As a result, more and more people choose to shop on E-commerce platforms. Besides bringing customers flexible and efficient shopping experiences, E-commerce companies also want to bring customers more personalized service, which may also bring the companies more profits. The recommendation system is an effective tool to do that.

Nowadays, every large-scale E-commerce company has its own recommender system. As one of the most popular E-commerce platforms, JD.com benefits a lot from its recommender system. During the ‘618’ in 2020, personalized recommendations shined, especially the "smart store", which realized the personalized distribution of event venues, which not only brought a significant increase in GMV, but also greatly reduced labor costs, and greatly improved traffic efficiency and user experience, So as to achieve a win-win situation for businesses and users.

2. Challenges

With the development of machine learning and deep learning algorithms, lots of methods are applied to recommender systems of E-commerce. Benham O. applied collaborative filtering clustering. [1] It's a method of using a co-occurrence matrix or similarity matrix to produce recommendations. Sarwar, Badrul M., et al. (2002) presented a clustering-based algorithm, which can adapt to the large-scale dataset. [2] Grbovic (2018) raised the use of embedding of users' click data to make real-time personalized search ranking[3]. Wu et al. (2020) proposed a method of combining user personas with the recommendation system, based on user portrait can meet the differentiated needs of different users[4].

JD.com's recommender systems are also improving with the development of recommender algorithms. In the beginning, JD.com used a simple association recommendation process to the personalized recommendation, which recommended similar products in users' buying process. Then, JD gradually transitioned to scene intelligence recommendation. The transition from related and similar product recommendations to multi-feature, multi-dimensional, user real-time behavior, and comprehensive intelligent recommendation based on user scenarios.[5]

All the methods above can be adapted into some datasets. Thus, the challenge for us is to find a better algorithm to fit our JD data. In other words, we want to build a proper recommendation system for the JD dataset.

3. Data Description

We collect data from MSOM and JD.com. JD.com and the MSOM society are partnering to offer members access to JD.com Transaction level data to encourage them to conduct data-driven research. This dataset contains several CSV files, we focus on three main: JD_order.csv, JD_user.csv, and JD_sku.csv.

JD_order.csv contains the main transaction information, such as order_ID, user_ID and sku_ID, and quantity, all of which are the core data we need to train our model.

	order_ID	user_ID	sku_ID	quantity
0	d0cf5cc6db	0abe9ef2ce	581d5b54c1	1.0
1	9d74489696	0abe9ef2ce	38d636d2a6	1.0
2	9d74489696	0abe9ef2ce	6717b7c979	1.0
3	e0f5386d87	0b07cae293	589c2b865b	1.0
4	252f6bbde1	010dc4d911	4fe57b6fb2	1.0

```
RangeIndex: 59736 entries, 0 to 59735
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   order_ID  59736 non-null  object
 1   user_ID   59736 non-null  object
 2   sku_ID    59736 non-null  object
 3   quantity  59736 non-null  float64
dtypes: float64(1), object(3)
memory usage: 1.8+ MB
```

JD_user.csv has basic information about each user: age, gender, education level, city level, marital status, etc. we use this dataset to build user portraits.

	user_ID	user_level	first_order_month	plus	gender	age	marital_status	education	city_level	purchase_power
0	000089d6a6	1	2017-08	0	F	26-35	S	3	4	3
1	0000bc018b	3	2016-06	0	F	>=56	M	3	2	3
2	0000d0e5ab	3	2014-06	0	M	26-35	M	3	2	2
3	0000f81d1b	1	2018-02	0	F	26-35	M	2	3	2
4	00012bb423	4	2008-11	1	F	26-35	M	4	1	2

```
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   user_ID            50000 non-null  object
 1   user_level         50000 non-null  int64
 2   first_order_month  50000 non-null  object
 3   plus               50000 non-null  int64
 4   gender             50000 non-null  object
 5   age                50000 non-null  object
 6   marital_status     50000 non-null  object
 7   education          50000 non-null  int64
 8   city_level         50000 non-null  int64
 9   purchase_power     50000 non-null  int64
dtypes: int64(5), object(5)
memory usage: 3.8+ MB
```

JD_sku.csv is used to provide more information related to the order.

	sku_ID	type	brand_ID	attribute1	attribute2	activate_date	deactivate_date
0	a234e08c57	1.0	c3ab4bf4d9	3.0	60.0	NaN	NaN
1	6449e1fd87	1.0	1d8b4b4c63	2.0	50.0	NaN	NaN
2	09b70fcd83	2.0	eb7d2a675a	3.0	70.0	NaN	NaN
3	acad9fed04	2.0	9b0d3a5fc6	3.0	70.0	NaN	NaN
4	d66bddc96f	2.0	4f65703579	-	100.0	NaN	NaN

```
RangeIndex: 3710 entries, 0 to 3709
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   sku_ID           3710 non-null   object
 1   type             3590 non-null   float64
 2   brand_ID         3590 non-null   object
 3   attribute1       3590 non-null   object
 4   attribute2       3590 non-null   object
 5   activate_date    271 non-null    object
 6   deactivate_date  67 non-null     object
dtypes: float64(1), object(6)
memory usage: 203.0+ KB
```

4. Methods

We implement 4 models to compare: two basic models, see 4.1 and 4.2, and two advanced models, see 4.3 and 4.4.

4.1. Simple Linear Model

This model uses Product and User as input, then through the embedding layer by the same dimension. This model uses the Order's quantity (the relationship between Product and User) as the target value.

It is worth noting that: The first three models use the same input and output, the difference is in the construction of the model. The fourth model uses totally different data input and should be seen as an independent part.

4.2. Simple Linear Model that Introduced Bias

The first model does not explicitly take into account the bias that a user might have in giving consistently every product he buys. In the second model, we introduced bias in the embedding layer. Basically, the second model is better than the first.

4.3. Neural Collaborative Filtering

The third model we use an advanced function: Neural Collaborative Filtering(NCF). And we use a multi-layer perceptron (MLP) to learn the user-item interaction function. And we present a new neural matrix factorization model, which ensembles MF and MLP under the NCF framework; it unifies the strengths of linearity of MF and non-linearity of MLP for modeling the user-item latent structures.[6]

4.3.1 Generalized Matrix Factorization (MF)

Under the NCF framework, MF can be easily generalized and extended. Due to the one-hot encoding of user_ID of the input layer, the obtained embedding vector can be seen as the latent vector of the user (item). Let the user latent vector $\mathbf{P}_u$ be $\mathbf{P}^T\mathbf{v}_u^U$ and

item latent vector $\mathbf{q}_i$ be $\mathbf{Q}^T\mathbf{v}_i^I$. Then the mapping function of the first neural CF layer as:

$$\phi_1(\mathbf{p}_u, \mathbf{q}_i) = \mathbf{p}_u \odot \mathbf{q}_i,$$

where $\odot$ denotes the element-wise product of vectors. We then project the vector to the output layer:

$$\hat{y}_{ui} = a_{out}(\mathbf{h}^T(\mathbf{p}_u \odot \mathbf{q}_i)),$$

where a_{out} and $\mathbf{h}$ denote the activation function and edge weights of the output layer, respectively. Intuitively, if we use an identity function for a_{out} and enforce $\mathbf{h}$ to be a uniform vector of 1, we can exactly recover the MF model. Under the NCF framework, MF can be easily generalized and extended.

4.3.2 Multi-Layer Perceptron (MLP)

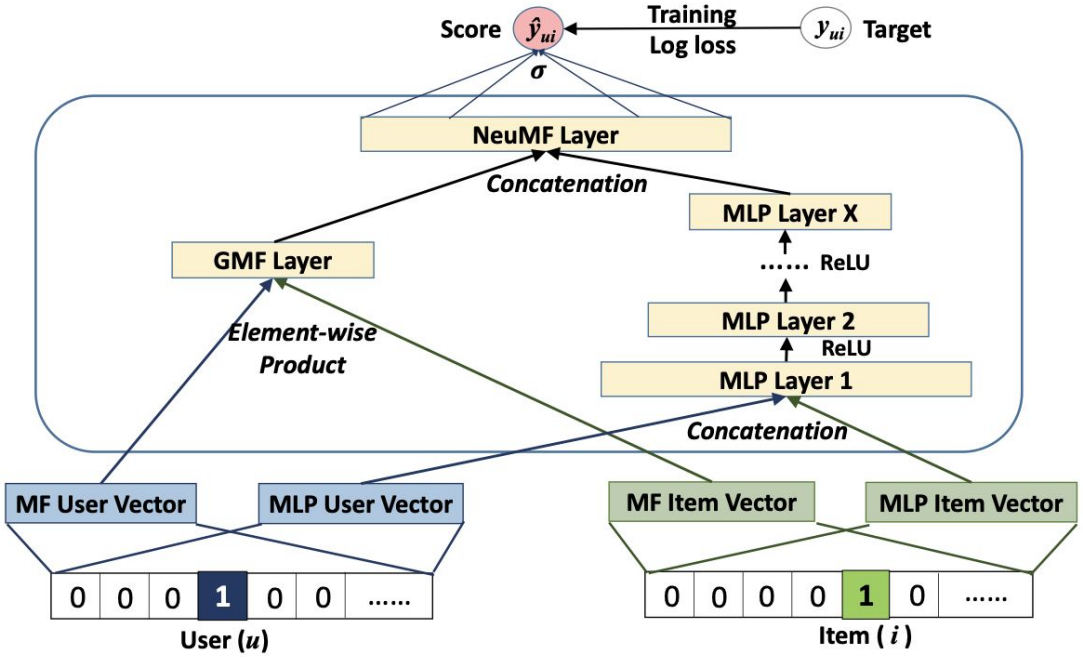
Since NCF adopts two pathways to model users and items, it is intuitive to combine the features of two pathways by concatenating them. However, simply a vector concatenation does not account for any interactions between user and item latent features, which is insufficient for modelling the collaborative filtering effect. To address this issue, this model adds hidden layers on the concatenated vector, using a standard MLP to learn the interaction between user and item latent features. More precisely, the MLP model under our NCF framework is defined as:

$$\mathbf{z}_1 = \phi_1(\mathbf{p}_u, \mathbf{q}_i) = \begin{bmatrix} \mathbf{p}_u \\ \mathbf{q}_i \end{bmatrix},$$
$$\phi_2(\mathbf{z}_1) = a_2(\mathbf{W}_2^T \mathbf{z}_1 + \mathbf{b}_2),$$
$$\ldots\ldots$$
$$\phi_L(\mathbf{z}_{L-1}) = a_L(\mathbf{W}_L^T \mathbf{z}_{L-1} + \mathbf{b}_L),$$
$$\hat{y}_{ui} = \sigma(\mathbf{h}^T \phi_L(\mathbf{z}_{L-1})),$$

where $\mathbf{W}_x$, $\mathbf{b}_x$, and a_x denote the weight matrix, bias vector, and activation function for the x-th layer's perceptron, respectively.

4.3.3 Fusion of GMF and MLP

Now we have two instantiations of NCF — MF that applies a linear kernel to model the latent feature interactions, and MLP that uses a non-linear kernel to learn the interaction function from data. Then fuse MF and MLP under the framework:



Let GMF and MLP share the same embedding layer, and then combine the outputs of their interaction functions. However, sharing embeddings of MF and MLP might limit

the performance of the fused model. For example, it implies that GMF and MLP must use the same size of embeddings; for datasets where the optimal embedding size of the two models varies a lot, this solution may fail to obtain the optimal ensemble.

To provide more flexibility to the fused model, NCF allows MF and MLP to learn separate embeddings, and combine the two models by concatenating their last hidden layer. The formulation of which is given as follows:

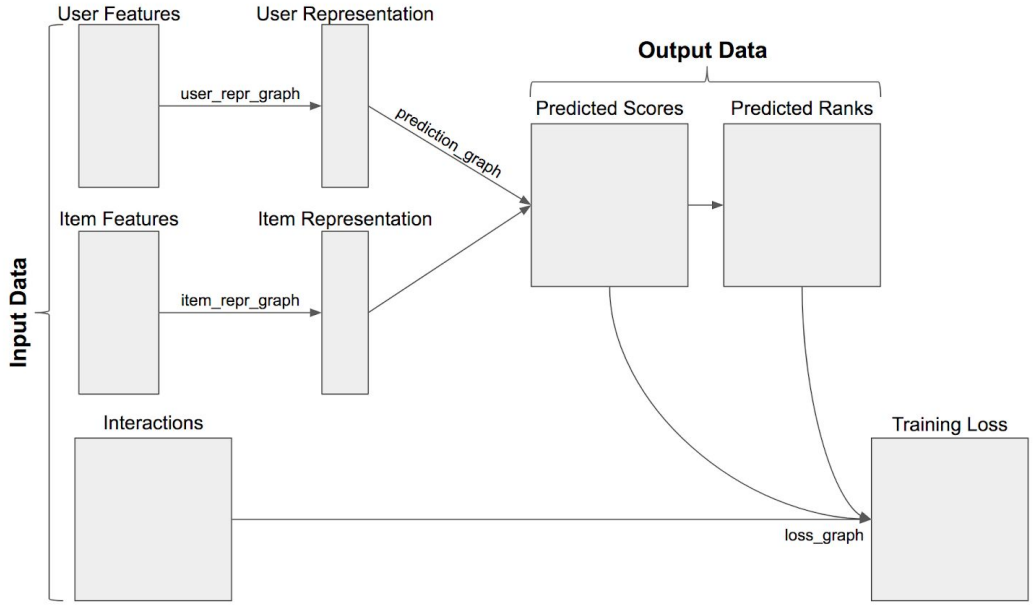
$$\begin{aligned}
\phi^{GMF} &= \mathbf{p}_u^G \odot \mathbf{q}_i^G, \\
\phi^{MLP} &= a_L(\mathbf{W}_L^T(a_{L-1}(...a_2(\mathbf{W}_2^T \begin{bmatrix} \mathbf{p}_u^M \\ \mathbf{q}_i^M \end{bmatrix} + \mathbf{b}_2)...)) + \mathbf{b}_L), \\
\hat{y}_{ui} &= \sigma(\mathbf{h}^T \begin{bmatrix} \phi^{GMF} \\ \phi^{MLP} \end{bmatrix}),
\end{aligned}$$

where **p**Gu and **p**Mu denote the user embedding for MF and MLP parts, respectively; and similar notations of **q**Gi and **q**Mi for item embeddings.

4.4. TensorRec

A TensorRec recommender system consumes three pieces of input data: user features, item features, and interactions. Based on the user/item features, the system will predict which items to recommend. The interactions are used when fitting the model: predictions are compared to the interactions and a loss is calculated, which the system learns to decrease.

Here is the structure of the Tensorrec system.



One way we can configure the TensorRec system is by changing the loss graph. The loss graph takes in predictions and interactions and calculates a loss that the system will try to decrease as it learns. By default, TensorRec uses RMSE (root mean square error) as the loss graph. The TensorRec also provides a loss graph called WMRB-weighted margin-rank batch, which is used to use in ranking. The loss function to incur "rank weighted" loss as follows:

$$L^{wmrb}(x, y) = \Phi^{wmrb}(r_y) = \log(r_y + 1)$$

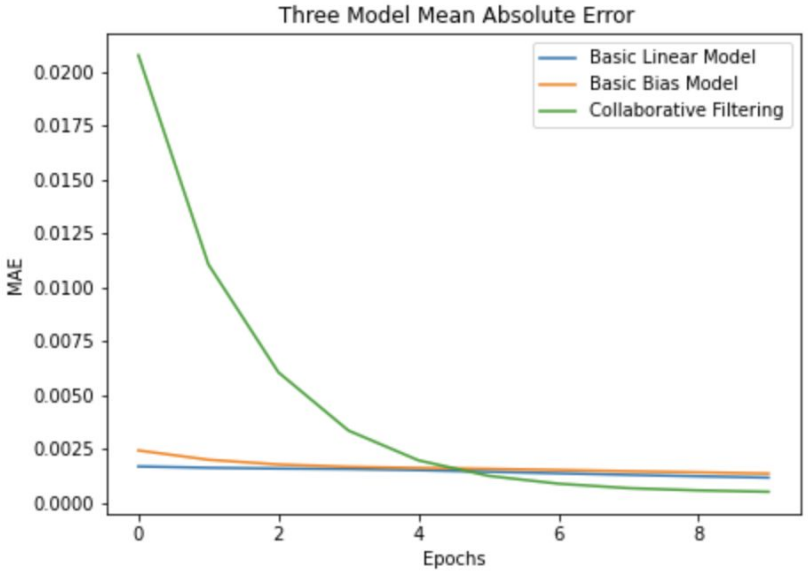
Where x denotes a user, y an item, and Y the entire item set. y_x denotes items interacted by user x. $\bar{\mathbf{y}}_x \equiv \mathbf{Y} \backslash \mathbf{y}_x$ is the irrelevant item set. $f_{y(x)}$ denotes the model score. The rank of item y is defined as:

$$r_y = rank_y(f, x, \mathbf{y}) = \sum_{\bar{y} \in \bar{\mathbf{y}}} \mathbf{I}[f_y(x) \leq f_{\bar{y}}(x)]$$

5. Results

5.1. Two basic and NCF model

These three models use the same input data. So, it can be compared together:



After 10 iterations, as we can see from the above chart, the NCF model has the best performance in mean absolute error(MAE). Then we use test data to make predictions.

Because we use a quantity of an order of a single user as the target value, the prediction is the (quantities) possibilities of a single product that the user may buy. It means very little. We need to explain the model further. So, we introduced K-means that categorize users. Each user group will have a bunch of products' quantities score, then ranking these products' quantities score, picking up the TOP10:

User_Cluster	TOP 1	TOP 2	TOP 3	TOP 4	TOP 5	TOP 6	TOP 7	TOP 8	TOP 9	TOP 10
0	60	100	35	441	648	579	945	171	2903	487
1	646	665	643	1584	576	441	1230	171	591	945
2	441	945	28	1779	196	1174	1475	1121	2459	2783
3	60	187	28	403	384	1475	99	2719	217	382
4	995	217	526	2787	1257	3166	423	2825	2973	1886
5	99	2790	28	995	2195	93	118	1088	2163	1917
6	1038	441	241	2968	285	201	196	2698	28	745
7	441	2589	127	99	2420	118	2235	153	1445	1734
8	643	441	171	241	1088	384	196	1600	3219	327
9	60	945	171	579	1618	93	591	2872	1936	52

The number in the above chart means a kind of product(sku_ID). We divided the users into 10 groups, and each group selected the top 10 products, then visualize:

	sku_ID	Rank_Score	User_Cluster
0	60	0.033830	0
1	100	0.024814	0
2	35	0.019971	0
3	441	0.019942	0
4	648	0.015910	0
5	579	0.015459	0
6	945	0.014476	0
7	171	0.014301	0
8	2903	0.014002	0
9	487	0.013276	0
10	646	0.039242	1
11	665	0.027386	1
12	643	0.025336	1
13	1584	0.019296	1
14	576	0.018365	1
15	441	0.017723	1
16	1230	0.017119	1
17	171	0.016363	1
18	591	0.015276	1
19	945	0.014605	1
20	441	0.017484	2
21	945	0.015391	2
22	28	0.014875	2
23	1779	0.013500	2
24	196	0.012277	2
25	1174	0.011578	2
26	1475	0.011569	2
27	1121	0.011478	2
28	2459	0.011426	2
29	2783	0.011274	2

5.2. TensorRec

TensorRec uses a different strategy to train the model. So, the output is different from the above three models.

user_ID sku_ID	000089d6a6	0000bc018b	0000d0e5ab	0000f81d1b	00012bb423	00015ff032	0001aa7059	0001bbdc89	0001dc70f4	0001f75444	...	1fb813187a	1f
a234e08c57	0.815756	0.825175	0.838938	0.806597	0.842491	0.846644	0.831375	0.813558	0.843178	0.818321	...	0.815971	
6449e1fd87	1.001001	1.009686	1.024361	0.989858	1.027873	1.031757	1.016084	0.997624	1.027195	1.003403	...	1.000444	
09b70fcd83	1.130153	1.136776	1.148941	1.122919	1.143696	1.144240	1.140505	1.127206	1.138404	1.131744	...	1.129301	
acad9fed04	1.130153	1.136776	1.148941	1.122919	1.143696	1.144240	1.140505	1.127206	1.138404	1.131744	...	1.129301	
d66bddc96f	1.216511	1.222239	1.240444	1.192144	1.260630	1.268553	1.232501	1.207869	1.263208	1.218798	...	1.213016	
...	...	...	...	...	...	...	...	...	...	...	...	...	
a8d3711371	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	
35fecca52d	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	
52cf82af76	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	
6dd03209f4	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	
3dd77315b0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	

3710 rows × 49712 columns

As we use 3710 items and 49712 users in the training, the output of the model is a table with 3710 rows and 49712 columns.

user_ID	top1	top2	top3	top4	top5
000089d6a6	7dc1fc309a	290f73077d	64e636ef58	20bffe9623	195ee43056
0000bc018b	11fe993823	4d4c737942	12e5f27006	147de61a88	439a5873bc
0000d0e5ab	b206d2f6d9	cac9171db0	fc738c9fc5	39c080d778	2ae88c6593
0000f81d1b	290f73077d	195ee43056	20bffe9623	64e636ef58	7dc1fc309a
00012bb423	49ad890e8e	831ad6e7e1	468088f3e5	e19067cdc2	11fe993823
...	...	...	...	...	...
1fb9c5324b	214818b3d7	8a2e72367f	16111b2973	49ad890e8e	fd187cb184
1fb9da04c7	64e636ef58	7dc1fc309a	20bffe9623	290f73077d	195ee43056
1fba172502	290f73077d	20bffe9623	64e636ef58	195ee43056	7dc1fc309a
1fba3a8b80	439a5873bc	8a2e72367f	16111b2973	49ad890e8e	f07d2373fa
1fba719c6e	20bffe9623	290f73077d	195ee43056	7dc1fc309a	64e636ef58

49712 rows × 5 columns

Then, we output the TOP5 sku_ID for each user. We use recall at k to evaluate the model. Recall at k is the proportion of relevant items found in the top-k recommendations. Mathematically recall@k is defined as follows: Recall@k = (# of recommended items @k that are relevant) / (total # of relevant items). We use k as 1500 for the 3710 items. The Recall at @k for the training data is 0.97 and the testing data is also 0.97.

6. Conclusion

In this project, we use four recommendation systems methods: simple linear model, simple linear model introduced bias, neural collaborative filtering, and Tensorrec. The first three models can predict the scores of each user in buying one specific item. Together with K-means clustering, we can get the rank of recommending items for every cluster. The tensorrec can give us scores of every user with every item. And we can directly get the rank of recommending items for each user.

In the future, we can try to use neural collaborative filtering together with Tensorrec. With two results, we can use boosting to vote the final results for users. And we shall need more data to feed a better recommendation system. And nowadays, the recommendation system is developing increasingly under the framework of deep neural networks, and it has been applied in various e-commerce platforms based on the recommendation system of each user. The most important task now is to collect more user data to solve the overfitting problem raised in the NCF model.

Reference

[1] Benham, Oliver. "eCommerce Recommendation Systems: A Collaborative Filtering Approach."

[2] Sarwar, Badrul M., et al. "Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering." *Proceedings of the fifth international conference on computer and information technology*. Vol. 1. 2002.

[3] Grbovic, Mihajlo, and Haibin Cheng. "Real-time personalization using embeddings for search ranking at Airbnb." *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018.

[4] Wu, Tiantian, et al. "Research on Recommendation system based on user portrait." *2020 IEEE I*nternational Conference on Artificial Intelligence and Information Systems (ICAIIS). IEEE, 2020.

[5] https://cloud.tencent.com/developer/article/1039697

[6]He, Xiangnan, et al. "Neural collaborative filtering." Proceedings of the 26th international conference on the world wide web. 2017.

[7] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J. Smola, and How Jing. 2017. Recurrent Recommender Networks. In Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM '17). Association for Computing Machinery, New York, NY, USA, 495–503. DOI:https://doi.org/10.1145/3018661.3018689

[8] Liu, Kuan, and Prem Natarajan. "Wmrb: Learning to rank in a scalable batch training approach." arXiv preprint arXiv:1711.04015 (2017).

[9]https://towardsdatascience.com/getting-started-with-recommender-systems-and-tensorrec-8f50a9943eef

[10]He, Xiangnan, et al. "Neural collaborative filtering." Proceedings of the 26th international conference on world wide web. 2017.