OTTO – Multi-Objective Recommender System

Kuangshi Ai, Junhao Shi
School of Data Science, School of Computer Science, Fudan University
ID: 20307140059, 20307140008
20307140059@fudan.edu.cn, 20307140008@fudan.edu.cn

January 8, 2023

Abstract

With the rapid development of e-commerce platform, it becomes increasingly important to build efficient recommmender systems, which not only brings customers nicer online shopping experience, but significantly increase the profit of platforms. In this paper, based on a real-world e-commerce dataset established by MIT in a kaggle competition, we build a session-based recommender system consists of various models with different approaches, ranging from simple item Collaborative Filtering to a co-visitation matrix based Candidate Reranking Model. We also adapt our models to allow training on GPU, achieving acceptable single model training time and satisfying scores. Notably, with voting ensemble technique, our work wins a silver medal in the corresponding kaggle competition.

1 Introduction

As online market mounts, shoppers have their pick of millions of products from large retailers. While such variety may be impressive, having so many options to explore can be overwhelming, resulting in shoppers leaving with empty carts. This neither benefits shoppers seeking to make a purchase nor retailers that missed out on sales. With the rapid development of artificial intelligence, retailers set their sights on recommender system.

Current recommender system consist of various models with different approaches. However, one primary problem is that no single model exists that can simultaneously optimize multiple objectives. Based on given full e-commerce session information , we build a multi-objective recommender system which predicts e-commerce clicks, cart additions, and orders.

In this paper, we build a multi-objective recommdender system using different recommend alhorithms, achieving 0.578 correctness in the corresponding kaggle competition, winning a silver medal according to the current leaderboard.

2 Dataset

We choose 'OTTO Recommender Systems Dataset' as our dataset, which also serves as dataset for kaggle competiton 'OTTO-Multi-Objective Recommender System' [1] This dataset contains 12M real-world anonymized user sessions, over 220M events (consiting of clicks, carts and orders) and 1.8M unique articles in the catalogue. Data are stored in the following format:

- sessions - the unique session id
- events - the time ordered sequence of events in the session
  - aid - the article id (product code) of the associated event
  - ts - the Unix timestamp of the event
  - type - event type

2.1 Basic Statistics

The statistics offered by Table 1 below illustrate that most of the events are "clicks", only a small proportion of them will lead to "carts", a smaller proportion leading to "orders", eventually purchasing the item.

From Table 2 and Table 3 it is notable that customers tend to make several events when online shopping, which suggests "clicks" and "carts" are made before "orders"(if made).By comparison, items' popularity varies greatly. While having an average of 116.79 visits, 50% of all items are visited less than 20 times, indicating that popular items are chosen much frequently.

Table 1: Statistics Overview

Dataset	**sessions**	**items**	**events**	**clicks**	**carts**	**orders**	**Density**
Train	12.899.779	1.855.603	216.716.096	194.720.954	16.896.191	5.098.951	0.0005
Test	1.671.803	TBA	TBA	TBA	TBA	TBA	TBA

Table 2: Events per session

#events per session	**mean**	**std**	**min**	**50%**	**75%**	**90%**	**95%**	**max**
Train	16.80	33.58	2	6	15	39	68	500
Test	TBA	TBA	TBA	TBA	TBA	TBA	TBA	TBA

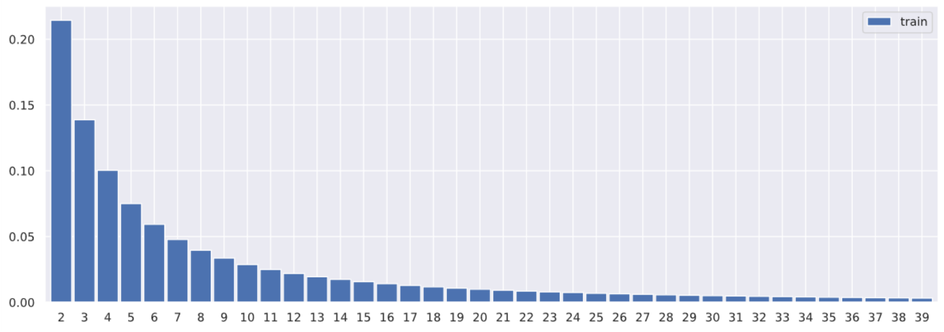


Figure 1: Events per session histogram (90th percentage)

Table 3: Events per item

#events per item	**mean**	**std**	**min**	**50%**	**75%**	**90%**	**95%**	**max**
Train	116.79	728.85	3	20	56	183	398	129004
Test	TBA	TBA	TBA	TBA	TBA	TBA	TBA	TBA

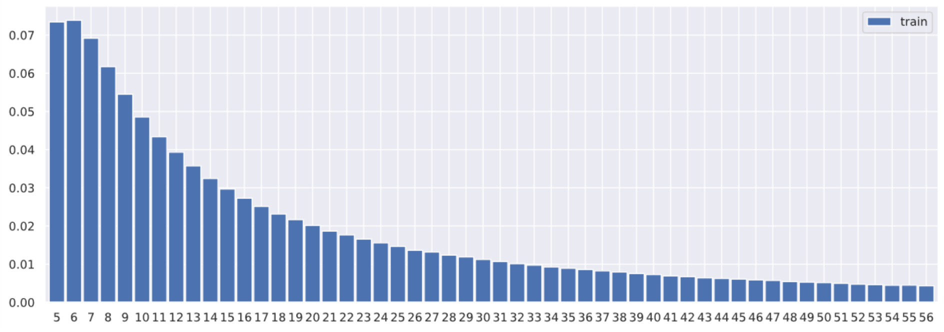


Figure 2: Events per item histogram (90th percentage)

2.2 Evaluation

We use Recall@20 as evaluation metrics for each action type, and the three recall values are weight-averaged:

$$score = 0.10 \cdot R_{clicks} + 0.30 \cdot R_{carts} + 0.60 \cdot R_{orders}$$

where R is defined as:

$$R_{type} = \frac{\sum_{i=1}^{N} |\{\text{predicted aids}\}_{i,type} \cap \{\text{ground truth aids}\}_{i,type}|}{\sum_{i=1}^{N} \min(20, |\{\text{ground truth aids}\}_{i,type}|)}$$

2.3 Data processing

2.3.1 Missing Values & Anomalies

No data is found missing in this datasaet and no anomaly is found , all user events are provided wholly, following the "click - cart - order" pattern.

2.3.2 Reformat & Compression

Though the format provided by primary dataset is vivid and clear for understanding, this multi-level form may cause difficulties for system training. Meanwhile, the Unix timestamp takes too much memory for unnecessary information. For all sessions happen in three weeks, we minus all timestamp by a fixed number and convert them into datatype numpy int32. By implementing measures above, we compress the training data from 11.3GB into 1.68GB.

For better analysis, data are reformed into following format:

- sessions - the unique session id
- events - the time ordered sequence of events in the session
- aid - the article id (product code) of the associated event
- time - timestamp of the event (np.int32)
- type - event type

2.3.3 Data splitting

According to the description of the dataset, the train set consists of observations from 4 weeks, while the test set contains user sessions from the following week. Furthermore, train sessions are trimmed overlapping with the test period, as depicted in the following diagram, to prevent information leakage from the future:

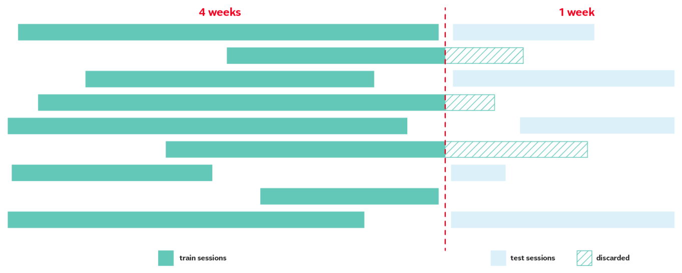


Figure 3: Train-Test-Split

For validation set, we use the truncated test sets from the training sessions provided by otto company and use this to evaluate their models offline.

To build a Leak free training of a two-stage recommender, we take examples from organizer's repo and split the train data of this competition (four weeks of train) into chunks of 3 weeks and the last week, getting:

- train1 - 3 weeks of train
- train2 - last week of original train THAT will be just like the test set

By using this setup, we can construct co-visitation matrices on train1 and train data from train2 while training a ranking model on train2, finally predicting on the competition test. For its validation set, we use similar approach that split train1 into two weeks of original train + the last week of train1. Then we can train our ranker on that week and validation on train2 from above.

2.4 Data analysis

In the dataset the item ids are anonymized, so we do not know which each item id refers to. However, with item matrix factorization we can convert the anonymus item ids into meaningful embeddings, then similar embeddings will be similar items. If we project the embeddings into the 2D plane (with TSNE in this case) and plot them, we can see clusters of similar items. When dots are close to each other in the top plot, then the items are similar, one cluster may be clothing and another cluster could be electronics, etc.

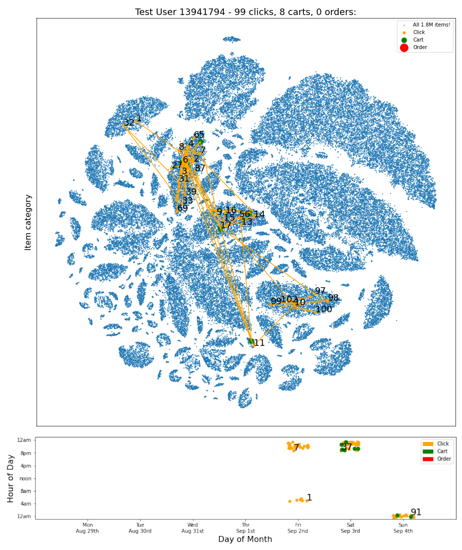
The top plots show the category of items interacted with and the bottom plot show the times of item interaction. In the top plots, by using the 2D plane of item embeddings, we can draw dots and lines showing the progression of each users' activity. If users stay in the same area then they are shopping similar items like clothing department. When a user's plot changes from one region of x-y-plane to another, then the user changes to a different category of item like moving from clothing shopping to electronics shopping.

The numbers in the plot are sorted by time. Number 1 is the first item a user interact with while number 10 is the 10th item etc. We only display some numbers so that the numbers are readable. Orange dots are clicks, green dots are carts, and red dots are buys. The 1.8 million blue dots are all unique items. We observe clusters of blue dots which represent different categories of items.

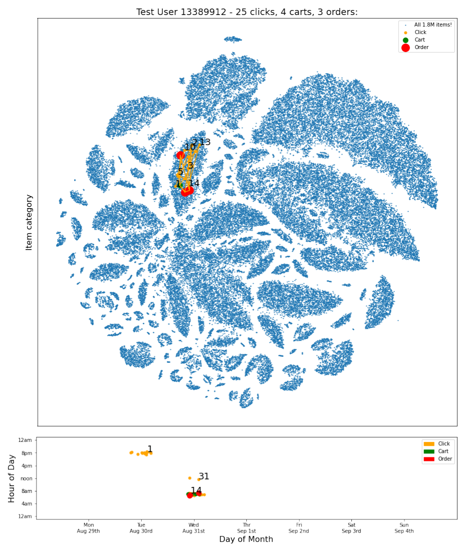
In the bottom plot, users' events are also plotted in time order, events happening close are presented in several clusters, while different actions are distinguished by colour.

We observe different types of users. Some users explore one category of item while other users explore a variety of item categories.This allows us to understand how users shop.

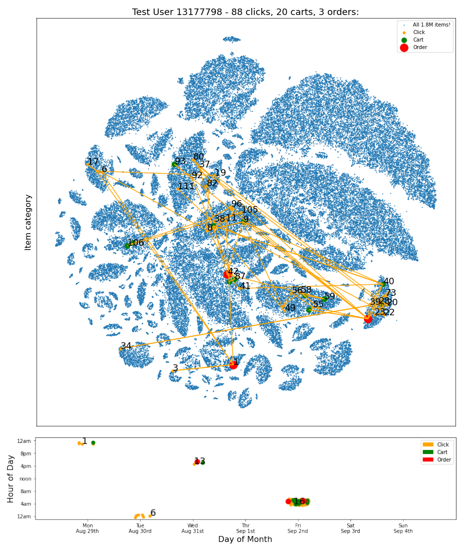
Examples of four random customers with more than 20 events are listed as follow:



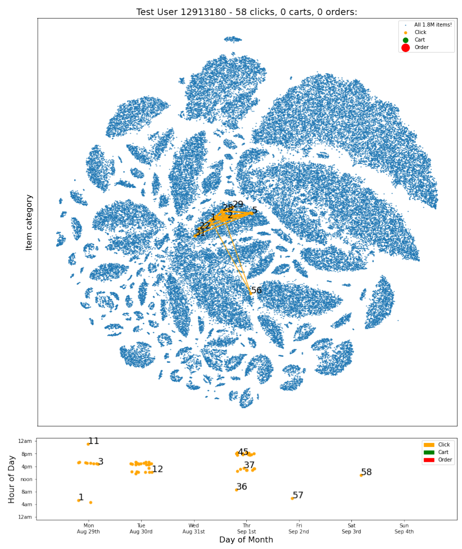
(a) user-13941794



(b) user-13941794



(c) user-13177798



(d) user-13941794

Taking (b) user 13941794 as an example, his/her shopping experience may be like:

On Tuesday Aug 30th, we see that user 13941794 first visits an item in the top-left cluster. They probably saw an ad somewhere on the internet and clicked it. The user gets interested in this category of item and make 11 more clicks, but didn't make any "carts" or "orders".

On Wednesday Aug 31st, user 13941794 returns to Otto website in the morning and continues viewing items in the previous category on the website. They had several more clicks , put items into their carts and made their first purchase. After placing an order, they continued viewing items in the same category and added two more orders. At noon, they came back and made two clicks in the same category, perhaps looking for the same item with a lower price ?

Viewing the user plots above, we can highlight following shopping patterns:

- The majority of "carts" and "orders" follows "clicks" and "carts" to the same item or similar items beforehand.
- Users' events can be divided into several clusters, both by item's scale or the time scale.
- If a user do "carts" in a cluster of items, he/she has an increased chance of doing "orders" and a slightly decreased chance of "clicks" in the same cluster.
- If a user gives "orders" to a specific item, he/she may continue viewing items in this cluster, possibly making "carts" or "orders", or just simply head away, concerning with items' category .
- If a user continues e-shopping after a time period, he/she may continue viewing items in the previous cluster. But if he/she clicks items in other clusters, the chance will become little.
- Users' shopping behaviour may vary on weekdays and weekends and might be affected by annual events or holidays

  (In this dataset, textbooks, schoolbags and other school-related items may spike in sales, since students are going back schools at September).

3 Methodology

We apply four different models: Item Collaborative Filtering, Matrix Factorization, Item2vec, and Candidate Reranking Model with hand-crafted rules to solve the problem. GPU training structures and voting ensemble technique are also used in our project.

3.1 Item Collaborative Filtering

Collaborative Filtering (CF) is widely used in recommender system, which is also the most traditional algorithm family in recommending. Owing to its limited complexity and not bad performances, the ItemCF algorithm is currently one of the most popular algorithms in the industry. The recommendation algorithms of Amazon, Netflix, and YouTube are all based on ItemCF. Thus, we choose item-based CF as a baseline for this OTTO recommending mission.

Similar to user-based CF, our algorithm analyzes the similarity between different items, and recommends the most likely interested 20 items to users based on their historical actions. Meanwhile, they have several differences which make ItemCF works better on our mission.

Suppose that we have a co-occurrence matrix $M \in \mathbb{R}^{m\times n}$, where m is the number of users, and n is the number of items, $M_{ij} = 1$ if user i conduct some action (including clicks, carts and orders, as a baseline, we do not distinguish different aid type for simplicity) on item j, otherwise $M_{ij} = 0$. In UserCF, we compute a user similarity matrix $U \in \mathbb{R}^{m\times m}$ based on M, where U_{ij} indicates the similarity between user i and user j, $i = 1, 2, \cdots m,\ j = 1, 2, \cdots m$.

However, due to the sparsity of M, for users who only have several purchasing or clicking actions, it is quite difficult to find similar users accurately. Thus, item-based CF may be a better choice. Instead of computing a huge user similarity matrix (in this mission we have more than two hundred million users in train set and six million users in test set), ItemCF computes an item similarity matrix $I \in \mathbb{R}^{n\times n}$. On the one hand, less time and space complexity are required, on the other hand, the problem of data sparsity has also been solved.

It is trivial to generate the co-occurrence matrix M, where we used the python module pandas to generate dataframe from the original .jasonl files. For simplicity, we only use data in test set and ignore the huge train set. The detailed code could be found in our notebook. One thing worth mentioning is that we use the python data structure dist to store sparse matrices, which takes much less space in RAM than storing full matrices.

With the co-occurrence matrix M, we choose a method akin to cosine similarity to generate the item similarity matrix $I \in \mathbb{R}^{n\times n}$.

$$I_{ij} = \frac{|N(i) \cap N(j)|}{\sqrt{|N(i)| \cdot |N(j)|}},\ i = 1, 2, \cdots n,\ j = 1, 2, \cdots n$$

where $|N(i)|$ represents the number of users (i.e. sessions) who conduct actions (including clicks, carts and orders) on item (i.e. aid) i.

Then for any item i, we can get a set $S(i, K)$ which contents the top K similar items as i. In this mission, set $K = 100$ and generate a similarity set $S(i, 100)$ for any item visited by the user. We also collect the global information and generate the most popular 20-item set overall $pop(20)$.

While predicting the 20 most likely next aids and item types, we do not distinguish different actions (which means attaching the same weights to them), and by sorting the similarity set $S(i, 100)$, we can get the most similar items without duplication. If the number of these items is greater than or equal to 20, the final result is the top 20 items. Otherwise, $pop(20)$ is used to compensate for those cases where the number of similar items is less than 20.

It is worth mentioning that in this baseline method, we only use the 6,928,123 pieces of data in test set, and predictions for clicks, carts and orders are all the same within a single session. However, the final result is pretty good, getting score 0.517 in the leaderboard (considering that the best LB score is 0.602 in Jan. 5th, 2023). This also helps us understand why ItemCF is still widely used in today's industry.

3.2 Candidate Reranking Model

Actually the train dataset is quite large, which consists of:

- 12,899,779 sessions
- 1,855,603 items
- 216,716,096 events

Therefore, it is unwise to simply dump the raw data into a model, which is less efficient. Thanks to other kagglers' discussion over the competition [2], we get to know a technique known as candidates generation, which is used by sites with billions of items like YouTube.

The following image gives an overview of the process:

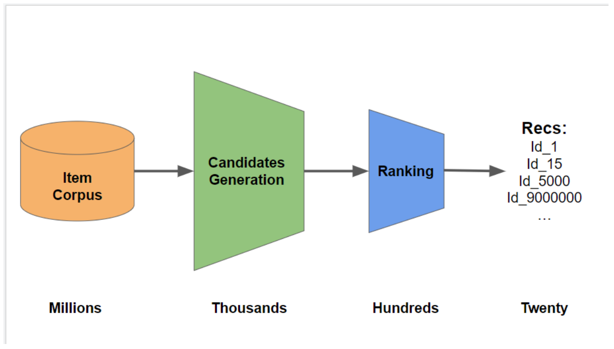


Figure 4: The process of Candidate Generation and Reranking

The first step is to generate candidates. With billions of input, we should figure out a cheap way to narrow the scope down. There are many possible criterions to use:

- previously purchased items
- overall most popular items
- similar items based on some sort of clustering technique
- similar items generated by co-visitation matrix

In the last model ItemCF, we use previously purchased items to generate item similarity matrix, and overall most popular items are also used as a backup for the main algorithm. In the following model, we will implement Matrix Factorization (MF) and Item2vec as two possible clustering techniques to generate candidates. Finally, with three kinds of co-visitation matrix, we deploy a Candidate Reranking model. After using these techniques, items for each session will be much fewer, from billions to thousands, then input them into a ranker model is possible.

The second step is to rerank and finally choose 20 possible to be predictions. Given the list of candidates, we use the following hand-crafted rules to select for us, these rules give priority to:

- most recently visited items
- items visited many times
- items in carts will be attached greater importance
- currently popular items

These hand-crafted rules can be modified with different parameter settings and weight assignments. In different cases, we have different priorities, which is further explained in our notebooks. This increases interpretability of the recommender system to some extend.

3.3 Matrix Factorization

The basic idea of Matrix factorization is a simple embedding method. Given the co-occurrence matrix $M \in \mathbb{R}^{m \times n}$ that we generate in ItemCF, where m is the number of users, and n is the number of items, the method learns:

- A user embedding matrix $U \in \mathbb{R}^{m \times d}$, where the ith row is the embedding for user i.
- An item embedding matrix $V \in \mathbb{R}^{n \times d}$, where the jth row is the embedding for item j.

such that $M = UV^T$, and d is the dimension of embedding layer.

With some basic knowledge of linear algebra, we may realize that eigen value decomposition and singular value decomposition able to use here. However, M is a sparse matrix and traditional numerical algorithms like QR algorithm bring heavy computing burden, a gradient descent style algorithm may work better here.

The embeddings are learned such that the product UV^T is a good approximation of the co-occurrence matrix M, i.e. the dot product $< u_i, v_j >$of the embeddings of user i and item j should be close to M_{ij}. To do this, minimize the sum of squared errors over all pairs of current pairs:

$$\min_{U,V} \sum_{(i,j)\in obs} \left(M_{ij} - u_i v_j^T\right)^2$$

where obs is the set of all observed entries. In order to prevent overfitting, we introduce a penalty entry for regularization, and the final objective function is:

$$\min_{U,V} \sum_{(i,j)\in obs} \left(M_{ij} - u_i v_j^T\right)^2 + \lambda(\|u_i\| + \|v_j\|)^2 \triangleq \min_{U,V} l(U,V)$$

And we can get the gradient descent direction by computing partial derivative. With stochastic gradient descent algorithm, only two partial derivatives are needed in each iteration.

$$\frac{\partial l}{\partial u_i} = 2\left(M_{ij} - u_i v_j^T\right) v_j - 2\lambda u_i$$
$$\frac{\partial l}{\partial v_j} = 2\left(M_{ij} - u_i v_j^T\right) u_i - 2\lambda v_j$$

where $i \in [m],\ j \in [n]$, both are uniformly distributed random variable generated in each iteration.

Algorithm 1 SGD for Matrix Factorization

```
Input: $U_0$: initial user embedding matrix; $V_0$: initial item embedding matrix; $M$: co-occurrence
    matrix; $\gamma$: learning rate; $\epsilon$: accuracy for termination $k \leftarrow 0$: step counter;
 1: repeat
 2:     $k \leftarrow k + 1$
 3:     generate two uniformly distributed random variables $i \in [m],\ j \in [n]$
 4:     compute gradient directions $g_{1k} = 2\left(M_{ij} - u_i v_j^T\right) v_j - 2\lambda u_i$, $g_{2k} = 2\left(M_{ij} - u_i v_j^T\right) u_i - 2\lambda v_j$;
 5:     update parameters $u_i \leftarrow u_i - \gamma g_{1k}$, $v_j \leftarrow v_j - \gamma g_{2k}$;
 6: until $\sum_{(i,j)\in obs} \left(M_{ij} - u_i v_j^T\right)^2 + \lambda(\|u_i\| + \|v_j\|)^2 < \epsilon$
Output: optimal $U^*, V^*$
```

In our code implementation, the process above is actually transparent with torch.nn, where nn.Embedding gives us an API to learn the factorization. U is the weight matrix between the input layer and the hidden layer, and V is the weight matrix between the hidden layer and the output layer.

Thanks to one kaggler's contribution [3, 4], we could make use of an efficient python module merlin to load data. Then comes the traditional machine learning training: define the model as a subclass of nn.Module, choose SparseAdam as an optimizer, choose nn.BCEWithLogitsLoss as loss function, seperate dataset into train set and validation set, keep training for several epochs and finally, model training ends.

Extract the weight of the embedding layer, we can get a hidden vector for any user or item. Then python module annoy is introduced for approximate nearest neighbor search. This way for any item (i.e. aid), we can find its nearest neighbor.

With MF, candidate generation is accomplished, then with several hand-crafted rules where we stress the importance of items added to carts, and most recently visited items also attached great importance, we could rerank those candidates and give the final 20 results. The most popular 20 items overall are also used as backup here.

3.4 Item2vec

Generally speaking, matrix factorization is a special kind of Embedding technique, encoding an object into a low-dimensional dense vector. In our mission, aid numbers can be regarded as sparse one-hot vectors with high dimensions, our goal is to find some Embedding vectors which are able to represent characteristics of different aids.

There are already many Embedding techniques to generate Embedding vectors, and recall layer techniques are widely used to compute item similarity and user similarity. After implementing rapid nearest neighbor research technology like locality-sensitive hashing, Embedding is a perfect choice for preliminary selection. Thus, we turn to word2vec model which is already mature in natural language processing.

In order to train a word2vec model, we need to find the implicit connection between different words. Two log-linear models [5] are taken into consideration: Continuous Bag-of-Words Model (CBOW) predicts the current word based on the context, while Continuous Skip-gram Model (Skip-gram) tries to maximize classification of a word based on another word in the same sentence.

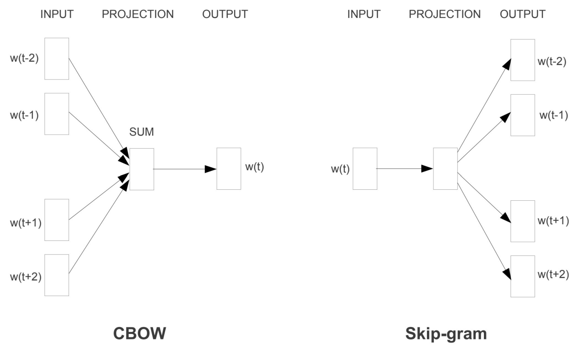


Figure 5: The model architectures of CBOW and Skim-gram

Take Skip-gram as an example, suppose that we have a slicing window whose length is $2c + 1$, then we fetch a sentence (i.e. session in our mission) from the dataset, slicing the window from left to right. Each time we move, the training sample inside the slicing window is renewed.

As any word w_t determines its neighbors $w_{t+j},\ -c \leqslant j \leqslant c,\ j \neq 0$, based on Maximum Likelihood Estimation, our objective function should be joint PDF of conditional probabilities $P(w_{t+j} \mid w_t)$:

$$\max \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leqslant j \leqslant c,\ j \neq 0} \log P(w_{t+j} \mid w_t)$$

The only problem here is how to define the conditional probabilities. It is natural to use softmax function to generate probability distributions, and as our aim is to represent a word w with a vector V_w, to give the similarity between two words with a dot product, we have the following defination:

$$P(w_O \mid w_I) = \frac{\exp\left(V_{w_O} V_{w_I}^T\right)}{\sum_{w \in W} \exp\left(V_w V_{w_I}^T\right)}$$

where w_O is the output word (i.e. w_{t+j}), w_I is the input word (i.e. w_t), and V_w is the vector to represent the word w.

Similar to how we implement Matrix Factorization above, the lookup table of Word2vec could also be gotten from weight matrices of Embedding layers. The weight matrix from the input layer to the hidden layer is the representation for input words, while the one from the hidden layer to the ourput layer is the representation for output words.

Actually we do not bother to construct our own Skim-gram model (due to the limited time), but make use of python module gensim to train the word Embedding dictionary. And again, python module annoy takes the job for approximate nearest neighbor search. After candidate generation, the old logic for hand-crafted rules are used in reranking, finally the 20 predictions are given.

As we make use of a list of clicks, carts and orders conducted by users to generate word Embedding, in general sense, our method is a kind of Item2vec. However, there is not slicing window in Item2vec model, the objective function has the following form:

$$\frac{1}{K}\sum_{i=1}^{K}\sum_{j\neq i}^{K}\log P(w_j \mid w_i)$$

The only difference from Word2vec is that any two items in Item2vec are regarded having connection. Besides, there are many other methods turning items into vectors. For example, Deep Structured Semantic Models (DSSM) is an end-to-end model successfully applied by Facebook and Baidu, which has efficiently improved the performance of recommender systems.

3.5 Co-visitation Matrix

Here we imply another method for candidate generation: co-visitation matrix. As what RNN does, the probability of a word given the words that came before is estimated, and this is also our goal in this session. A co-visitation matrix counts the co-occurrence of two actions in close proximity. If a user bought A and shortly after bought B, we store these values together. Then we calculate counts and use them to estimate the probability of future actions based on recent history.

As suggested by kaggler Chris Deotte's brilliant work, we compute three co-visitation matrix for candidate generation:

- Co-visitation matrix of click/cart/order to cart/order with type weighting
- Co-visitation matrix of cart/order to cart/order called buy2buy
- Co-visitation matrix of click/cart/order to clicks with time weighting

This part of our work is based on the structure given by Chris, and it does improve the model performance significantly. In the second reranking part, we put forward our own hand-crafted rules varying from different scenarios. The hand-crafted rules give priorities to:

- Items previously in carts have weights 20 times larger than those in clicks and orders
- Events happening within one day are paired together
- Different action type is weighted differently in "Carts Orders" co-visitation matrix
- Recently happening events are weighted more in "Click" co-visitation matrix
- Recently clicks is more important in suggesting clicks than orders than carts
- Most popular items overall as backup

The detailed implementation can be found in our notebook. And this model performs best among all as a single model, its output and further exploration will be given in next section.

3.6 Training on GPU

To speed up our training process, we turn to GPU resources offered by kaggle platform. Thanks to Chris's work, we are able to adopt an intelligent GPU training structure. As GPU on kaggle platform only offers 16GB RAM, the 30GB CPU RAM is used as a cache between the disk and the GPU RAM. With limited available resources, Chris's brilliant work makes our training as efficient as possible. Below are some of the tricks to speed up computation:

- Use RAPIDS cuDF GPU instead of Pandas CPU
- Read disk once and save in CPU RAM for later GPU multiple use
- Process largest amount of data possible on GPU at one time
- Merge data in two stages. Multiple small to single medium. Multiple medium to single large.
- Write result as parquet instead of dictionary

The timestamps in the dataset are precise to seconds, stored as uint32 and the item types are stored as uint8, which reduces the space occupation as much as possible. Computing the three co-visitation matrices using RAPIDS cuDF on GPU is 30x faster than using Pandas CPU. It takes only three minutes to compute each co-visitation matrix. This way we can reduce the training time of co-visitation matrix based Candidate Reranking Model (cvm-CRM) to about an hour.

4 Results

4.1 Single Model Results

In this part, we give the final results of models mentioned above. As required by this kaggle competition, all results should be written into a .csv file and submitted to get an evaluation score. The test set will not be released until the end of the competition. Recall@20 is used as evaluation metrics for each action type, and the three recall values are weight-averaged:

$$score = 0.10 \cdot R_{clicks} + 0.30 \cdot R_{carts} + 0.60 \cdot R_{orders}$$

The following table contains execution time and result scores for each single model.

Table 4: Singe model execution time and scores

Model name	Execution time (s)	score
ItemCF	2207.3	0.517
Matrix Factorization	7604.9	0.493
Item2vec	2095.7	0.521
Co-visitation Matrix based CRM	3657.7	0.577

Item Collaborative Filtering. As a baseline method, 0.517 is relatively a nice score. Our code directly makes us of the original .jsonl data, where the timestamp is accurate down to the millisecond level and item type is represented by strings. This means heavy burden for CPU RAM and the dataloading process also takes much more time. Once turn to .parquet format simplified dataset, the execution time can be shortened by more than 100s.

Matrix Factorization. MF is the slowest and worst method among all. This is mainly to blame that the product of the user embedding matrix and the item embedding matrix UV^T converges to the co-occurance matrix M very slowly. In our notebook, the training lasts 4 epochs with learning rate 0.08, and it takes more than 2 hours. However, the TrainLoss only drops from 0.607 to 0.597, the accuracy only improves from 0.702 to 0.714. Though SGD does work well here, owing to the huge number of data and limited computing resources, the training time needed is unbearable. MF may not be a practical algorithm for large-scale recommending.

Item2vec. Compared with traditional Matrix Factorization, Item2vec takes much less time to train and has much better final results. The result is somehow interpretable. One thing interesting is that most parts of the results given by the two method is the same, but those slight differences significantly show that Embedding has much better characteristic-representing ability.

Co-visitation matrix. With co-visitation matrix to generate candidates and hand-crafted rules to rerank candidates, Candidate Reranking Model gives the best predictions. And with GPU computing and CPU RAM as cache, the execution time is also acceptable. In the submitted version of our notebook, we seperate the four-week train data into two parts: the first three weeks for train set and the last one week for validation set, which means that we have a labelled validation set to adjust our hyperparameters.

Hyperparameter tuning in Candidate Reranking Model. Based on our life experience, items previously in carts will be clicked often, and of course they are most likely to be included in orders. But what shocks us is that items in carts are so important that when we adjust the weights for clicks, carts and orders to 0.5 : 9 : 0.5, the final performance seems to be the best among all possible weight combination. Besides, the number of top items for three types generated from co-visitation matrices also affects final results a lot. This is similar to Attention Mechanism, which gives the maximum history information that our model can remember.

However, due to the limited computing resources that we have, we cannot make use of cross-validation to adjust hyper parameters. It takes about one hour to train the model and test it on validation set once. That is to say we may extract more possibility from this model with more time and computing resources.

4.2 Voting Ensemble with Weights

A good solution to improve the final results will be voting ensemble, which allows results attained from different method combine together. And voting ensemble with weights allows us to put more weight on predictions that got a better validation/LB score. This method can be extremely effective when more varied solutions are used in ensemble.

Table 5: Multi-model Ensemble results

Ensemble Models	Weight Assignment	score
cvm-CRM [0.577], MF[0.493], TDL[0.522]	1:0.4:0.55	0.517
cvm-CRM [0.577], Item2vec [0.521], ItemCF[0.517]	1:1:1	0.54
cvm-CRM [0.575], cvm-CRM [0.576], cvm-CRM [0.577]	1:2:3	0.577
cvm-CRM [0.575], cvm-CRM [0.576], cvm-CRM [0.577]	1:1:1	0.578
cvm-CRM [0.576], cvm-CRM [0.577], ensemble-CRM [0.578]	1:1:1	0.578

In theory, if we use results from varied methods, the voting ensemble technique should bring us more benefits. However, the first challenge that we met is the limited available resources. With each submissions file at over 5 million rows, each row containing 20 predictions, the 30GB RAM of kaggle CPU only allows us to combine three submissions at most, even with very memory efficient python module polars.

At our first trial, we tried to combine our best model co-visitation matrix based CRM (cvm-CRM) [0.577] and Matrix Factorization (MF) [0.493] with another different-styled model in a public notebook: Test Dataset Leak [6] (TDL) [0.522] by Tomoo Inubushi. With 1 : 0.4 : 0.55 weights assignment, the score of the ensemble submission is 0.534.

Then we use the top 3 submissions of our single model: co-visitation matrix based CRM, Item2vec and ItemCF. With 1 : 1 : 1 weights assigned to each result, the score of the ensemble submission is 0.54.

It seems as if we combine 1 good solution with 2 that are not that great by voting ensemble, the final result will not be better than the good solution. Thus, we also combine three different results generated by co-visitation matrix based CRM with different hyperparameter setting, whose scores are 0.575, 0.576, 0.577. With 1 : 2 : 3 weights assignment, the score is 0.577; while with 1 : 1 : 1 weights assignment, the score is 0.578!

It suddenly occurs to us that what if we use some matryoshka dolls trick. That is using the 0.578 output as an input, combining three submissions whose scores are 0.576, 0.577, 0.578. However, it does not work and still has score 0.578. But the voting ensemble technique indeed contributes to our final submission and helps us to win a silver medal (ranking 92, Jan. 7, 2023) in the kaggle competition!

5 Conclusion

In this paper we build a multi-objective recommdender system using different recommend alhorithms, including Item Collaborative Filtering, Candidate Reranking, Matrix Factorization, Item2vec and

Co-visitation Matrix. With voting ensemble strategy, we finally achieved a satisfying result. In the near future we will continue adapting this model, doing our best to achieve better scores in kaggle competiton.

5.1 About data leakage

Data leakage occurs when information from outside the training data is used to create the model, and can lead to overfitting and poor generalization to new, unseen data.

Take item recommendation as an example, if data leakage occurs in the data that you use to train the model, the model will be able to make very accurate predictions simply by memorizing the training data, rather than learning to identify the underlying patterns that lead to recommend. When you evaluate the model on new data, it will perform poorly because it has not actually learned to generalize from the training data. And that's the reason we need to build a leak-free training set.

In this competition, we are predicting test events occurring within a one week period.

When we predict events occurring in first day of test data, a desired situation is that we use the model trained on training set (past three weeks) to give our prediction. In the real situation, however, we can use events occurring in the next 6 days of test data which are from the future to make better predictions. Of course this leads to better prediction correctness, but similar situations will not happen in reality.

The solution might be rearranging the test sessions by the order of time and giving predictions sequentially, but this might be critical to implement for matrix factorization. Since the competition host allows the use of data leakage in the rules, we are not taking measures for this in our model.

Though data leakage is generally considered to be a bad thing, in item recommendation data leakage is acceptable or even desired. If the customer's purchase history is included in the training data, this could be considered data leakage because it is information that would not be available when making predictions for new customers. However, it may still be useful to include this information in the model because it can help the model make more accurate predictions for the specific customers in the training data. In this case, the goal of the model is not to generalize to new, unseen customers, but rather to make the best possible predictions for the specific customers in the training data.

5.2 Dive into data splitting

There are two important issues when splitting a dataset: how to split the validation and training sets, and whether to split a single dataset into multiple datasets. In projects involving the ranking models of computing advertisements and recommendation systems, we found that most customers use one of two typical methods for training: the T+1 training method and the daily incremental training method.

In this paper, in order to avoid data leakage, we choose to spilt data into the following format:

- train1 - 3 weeks of train
- train2 - last week of original train THAT will be just like the test set

Considering our computation resources, we fist choose to set the time length as a week. If we choose to divide the train data into blocks of days, the splitting result goes the same as the T+1 training method, which is used by the majority.The T+1 training method is shown in the following figure:

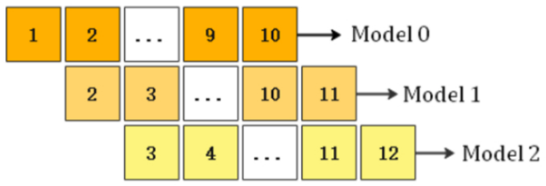


Figure 6: T+1 data splitting

As shown in the figure above, the T+1 training method has several elements: a full T days training set and the model is trained from scratch; the following day is used as the validation set; training based on a sliding window with a window size of 1 day; the online model is updated every day. Although the ranking task is not a strong time-dependent task like time series prediction, the boundary between the T days training set and the 1 day validation set in the most common T+1 training method is still divided according to the time order.

The 'T' here doesn't have a strict range, meanwhile '1' is neither strictly limited to 1 day. The T+1 splitting fits the standard of avoiding data leakage discussed above , though bringing other disadvantages as well. Compared with the splitting strategy we use in the model, its training time cost is roughly ten times bigger (21/2). Since using data leakage in the test set, we choose to keep on using the simpler strategy.

5.3 Possible future improvements

Due to the daily five submission restriction, we haven't fully adjust our weights in method of voting ensemble with weights, leaving possible improvement space.

Furthermore, several shopping patterns may be further mined and made use of:

- As is mentioned in Data analysis beforehand, the willingness to have further events differs according to items' categories.(People are likely to order more than one snacks/clothes during purchase, meanwhile the possibly for electronic gadgets is little) A possible solution is to label items according to clusters and specific shopping patterns (1 - high possibility, 0 - low possibility) These labels serve as additional variables in co-visitation training.
- For 'time' variable, current method uses it to show the sequence of users' events. For events having interns not longer than 12 hours, we tend to regard them as continuous. However, users' favourites for items or events(clicks/carts/orders) may also have relations with shopping time. Possible simple solution may be diving daily 24 hours into 3 blocks (0-8 midnight, 8-16 day, 16-24 night), a session is determined by the first event's timestamp. For three time periods we each train a model and finally combine them together.
- For holiday/annual event affections, we can imitate the thought of Attention, give the seasonal items a higher weight during specific period, while emphasizing daily products when it's not, giving seasonal items a punishment when out of season.

Moreover, voting ensemble may have a specified voting strategy. For example, results with high correctness may have high weights with all predictions, while those with low correctness may have fewer weights and only part of the predictions(top10,for example) may count in voting. For predicts in one result, weights can also vary according to priority.

Considering the limited computing resources we have, we didn't use large neural networks like RNN, CNN and Transformers(for it easily runs out of memory with such a large dataset) The main theory is to generate pairs of "user-item" and predict likely of click, cart, and order. For large-pretrained models (Transformers4Red,for example), we can still achieve good results after fine-tuning.

We will continue advancing our model before the kaggle competition dues.

5.4 General recommender system

The entire process of personalized Top-N recommendation contains three sub-stages[7]: the recall stage, ranking stage, and re-ranking stage. The recall stage refers to the process of selecting a small portion of possible recommended items from a large number of options. In this stage, the system considers the user's preferences and other information, and selects items relevant to the user from the database. In this part we will only discuss strategies of recall stage.

Different from only user-item relationships given in this dataset, in reality retailers get many other information of both users and items, including users' age, gender, residence, short/long-term shopping history, consumption level, items' price, category, current shoppping trend and so on.

Take Amazon as an example, Amazon's recommendation includes:

- personalized category recommendation
- Frequently bought items

- Your recently viewed items and featured recommendations
- Your browsing history
- Related to items you've viewed
- Customers who bought this item also bought
- A newer version of item bought
- Recommendations based on history orders
- Best-selling items
- (e-mail)Viewed category and best-selling brands
- (e-mail)other best-selling brands in the same category
- (e-mail)Bundle of viewed items
- (e-mail)Category of viewed items

According to Amazon's official technical blog[8],there are four basic recall strategies:

- Recall strategy based on hotness/popularity/rankings:

  It can be combined from two dimensions: the statistical period and the top categories of the item, usually using weighted scores, which can refer to the following formula:

  $$weighted\_rating = \frac{v}{v+m}R + \frac{m}{v+m}C$$

- Recall strategy based on item profiling:

  Item profiling refers to the portrayal of one aspect of the item itself. For example, the brand of the item, the price of the item, and the category to which the item belongs are all called item profiles. These profiles have obvious physical meanings and are called explicit profiles. If the embedding of the item ID is learned through a model, which embedding is called an implicit profile. From the perspective of explicit profiles and implicit profiles, recall strategies based on item profiling are naturally divided into: recall based on item content and recall based on the overall embedding of the item.

- Recall strategy based on collaborative filtering: (Used in this paper)

  This method is designed based on user behavior data only, for example, based on the User-Item matrix or the graph consisting of users and items, without any other features. It is essentially a matrix completion approach. Recall strategies based on collaborative filtering can be further divided into the following three types: collaborative filtering based on user / UserCF; collaborative filtering based on item / ItemCF; collaborative filtering based on model.

- Recall strategy based on user profiling:

  To Simplfy, user profiling is labeling users with tags. Recall based on user profiling is the key of personalized recommendation, because personalization is reflected through user profiling. User profiling includes the following three categories:

  - Basic information: Including Gender, age, residence, occupation, income level, etc.
  - Interest profiling: interest, including topics, keywords, entities, etc.
  - Behavioral profiling: List of item IDs for a user's recent N actions or actions over a certain period of time;Number of actions by a user in a week; user's spending level; user's click rate/conversion rate in a month, etc.

After specifying four basic strategy , next is paradigms of recall, which need to be considered from three dimensions: whether there are finer sub-strategies under a single recall strategy; whether the recall strategies are prioritized, and if they are prioritized, how to merge multiple recall strategies. Those strongly depend on the detailed task and the system's goal, which are not in the range of this discussion.

In a nutshell, although our model is far from actual application, we will continue to enhance our capabilities and remain confident that we will become qualified computer engineers in the future.

References

[1] Timo Wilm Philipp Normann, Sophie Baumeister. Otto-multi-objective recommender system. `https://github.com/otto-de/recsys-dataset`.

[2] Ravi Shan. Recommendation systems for large datasets. `https://www.kaggle.com/competitions/otto-recommender-system/discussion/364721`.

[3] NVIDIA Developer. Merlin dataloader. `https://github.com/NVIDIA-Merlin/dataloader`.

[4] Radek Osmulski. Matrix factorization [pytorch + merlin dataloader]. `https://www.kaggle.com/competitions/otto-recommender-system/discussion/366893`.

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv e-prints*, page arXiv:1301.3781, January 2013.

[6] Tomoo Inubushi. Test dataset is all we need? `https://www.kaggle.com/code/tomooinubushi/test-dataset-is-all-we-need/notebook`.

[7] Yuhui Liang. Recommendation system series: Overview of recommendation systems (part 1). `https://aws.amazon.com/cn/blogs/china/recommended-system-overview-of-recommended-system-series-part-1/?utm_campaign=New+message+4&utm_medium=email&utm_source=newsletter`.

[8] Yuhui Liang. Recommendation system series: In-depth discussion of the recall stage in recommendation systems. `https://aws.amazon.com/cn/blogs/china/in-depth-discussion-on-the-recall-stage-of-recommendation-system-of-recommendation-system-se`