WSM Final Project

WSM_Panda

許博堯（**110971005**）、盧奕潔（**109703060**）、
范佳琦（**109703027**）、田展禾（**109703037**）、
俞懿（**109208064**）

Introduction

For the final project this semester, we are participating in an e-commerce recommendation competition hosted by OTTO on the OTTO Kaggle Challenge website. The goal of this competition is to predict e-commerce clicks, cart additions, and orders. We build three multi-objective recommender systems based on previous events in a user session, using Tf-idf, XGBRanker, and Word2Vec Model.

TF-IDF

Model Introduction

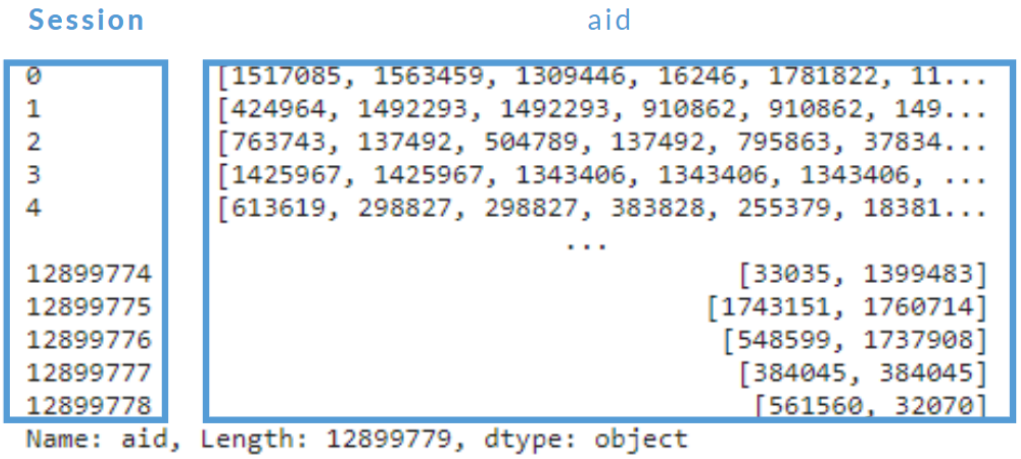
TF-IDF (Term Frequency - Inverse Document Frequency) is a handy algorithm that uses the frequency of words to determine how relevant those words are to a given document. It's a relatively simple but intuitive approach to weighting words, allowing it to act as a great jumping off point for a variety of tasks.

Parquet

- More appropriate to Data Analysis.
- Save data according to row.
- When the amount of data is large, it will be more efficient to use parquet.

Steps

1. After turning the train data into a form of parquet, groupby the session's aid.



Session	aid
0	[1517085, 1563459, 1309446, 16246, 1781822, 11...
1	[424964, 1492293, 1492293, 910862, 910862, 149...
2	[763743, 137492, 504789, 137492, 795863, 37834...
3	[1425967, 1425967, 1343406, 1343406, 1343406, ...
4	[613619, 298827, 298827, 383828, 255379, 18381...
	...
12899774	[33035, 1399483]
12899775	[1743151, 1760714]
12899776	[548599, 1737908]
12899777	[384045, 384045]
12899778	[561560, 32070]

Name: aid, Length: 12899779, dtype: object

2. Initially, we use Sklearn, however due to memory limit, we use Gensim Tfidf Model later instead.

```
dct = Dictionary(df_sess_split)  # fit dictionary

corpus = [dct.doc2bow(line) for line in df_sess_split] # convert corpus to BoW format

tfidf_model = TfidfModel(corpus)  # fit model
```

   (1) Build unique term by Gensim dic.
   (2) Transform corpus into bag-of-words model.
   (3) Put the corpus into Gensim TfidfModel, and calculate the tfidf value.

3. Build the query vector, and find the top score.
   (1) Read the test data and do the process as the same as train data.
   (2) Build the test set by dic, in order to match the aid in test data with the dic index and the number of actions.
   (3) Sort the aid according to the tf-idf value.

Rank and Submission

1. Find the top 20 popular aids according to the top biggest 20 tfidf values.
2. For every test session, we recommend products according to the total number of times that the items are either click, add to cart, order. The items with more time action means the more important.
3. If the number of recommended products from step 2 is less than 20, recommend the 20 popular products.

The score for this submission is 0.483.

Analysis and what we learned from this competition

- Parquet is a fast and useful storage format in data analysis. We also learned about pickels and other storage formats during the process.
- This model can be improved by considering the weight of different actions. Since click, add to cart and order can represent different importance of consumers.

Word2Vec

Model Introduction

Word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, the model can detect synonymous words or suggest additional words for a partial sentence. As the name implies, word2vec represents each distinct word with a particular list of numbers called a vector. The vectors are chosen carefully such that they capture the semantic and syntactic qualities of words; cosine similarity can indicate the level of semantic similarity between the words represented by those vectors.

Steps

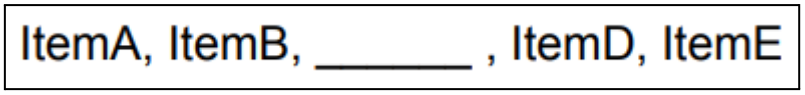
1. Grab the aids of train and test in session units, and organize them into a two-dimensional list as training data.
2. Put the training data into the Gensim Word2Vec Model.
3. Use Annoy to look for nearest neighbors in the embedding space.

Model training

In this model, we use CBOW technique. We regard aid as a word and predict the probability of current aid occurrence by former and latter aids. CBOW then produces an array for each aid, and the array represents the relationship of this aid with other aids.

CBOW

The bag-of-words model is a simplifying representation used in NLP and IR. In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. It is commonly used in methods of document classification where the (frequency of) occurrence of each word is used as a feature for training a classifier.
CBOW is faster than Skip-gram (predict the latter and former aid by current aid), it improves our training efficiency.



Rank and Submission

In order to search the nearest neighbor, we use the Annoy technique. (Initially we use gensim.most_similar(), however its execution time is too long. Therefore, we changed to using Annoy, which works the same but faster.) Then we calculate the similarity by using Euclidean distance.
The score for this submission is 0.521.

Annoy

Annoy is almost as fast as the fastest libraries. Besides, it also has the ability to use static files as indexes, this means one can share index across processes. Annoy also decouples creating indexes from loading them, so one can pass around indexes as files and map them into memory quickly. Another advantage of Annoy is that it tries to minimize memory footprint so the indexes are quite small.

Analysis and what we learned from this competition

- The scores of click, cart, order are the same because session type is not considered. We can improve the model by adding the information of type to the training data in the future.
- Because the context is considered, the score is relatively improved.

- We learned to train models by using transfer learning, and learned to stand on the shoulders of giants to improve our skills faster.

XGBRanker

Model Introduction

XGBoost is an optimized distributed gradient boosting library. It implements machine learning algorithms under the Gradient Boosting framework.
It supports Scikit-Learn Wrapper interface, which could be easily used as the other Scikit-Learn model.

Steps

1. Feature engineering
2. Train test split
3. Model training
4. Rank items

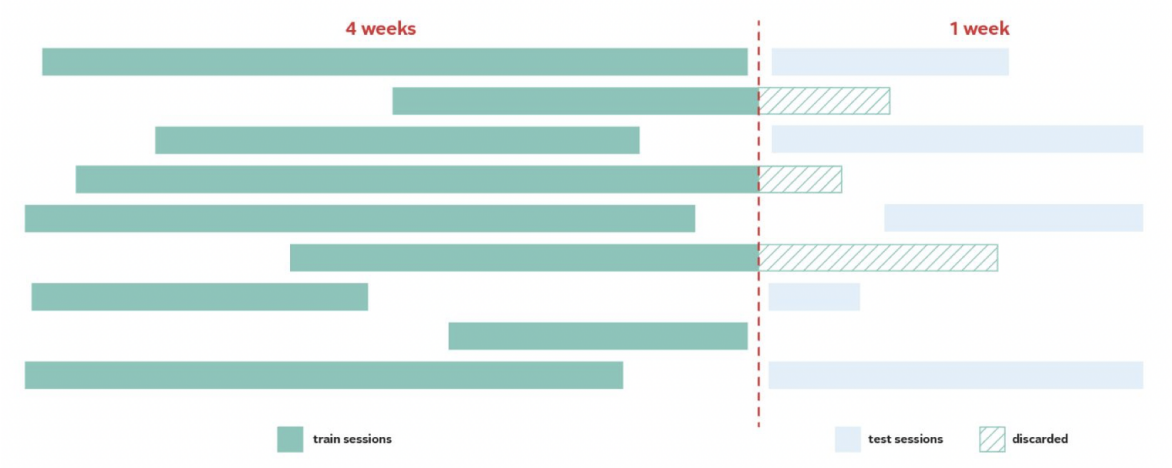
Feature engineering

Use the session, aid, timestamp, type columns to generate behavior features of session and aid:

- Features group by session
  - view_aid
  - click_cnt
  - cart_cnt
  - order_cnt
  - monday_action_cnt
  - tuesday_action_cnt
  - wednesday_action_cnt
  - thursday_action_cnt
  - friday_action_cnt
  - saturday_action_cnt
  - sunday_action_cnt
  - evening_action_cnt
- Features group by aid
  - viewed_session
  - clicked_cnt
  - carted_cnt
  - ordered_cnt
  - monday_action_cnt
  - tuesday_action_cnt
  - wednesday_action_cnt
  - thursday_action_cnt
  - friday_action_cnt
  - saturday_action_cnt
  - sunday_action_cnt
  - evening_action_cnt

Train test split

The objective of this competition is to predict the items in the future, so we choose the first 4 weeks as train data and the last 1 week as the test set, which should be an ideal splitting.



Model training

The default objective of XGBRanker is pairwise. NDCG takes into account both rank and relevance of retrieved documents. It should give the best results, as information about ranking is fully exploited and the NDCG is directly optimized. Therefore, the objective is set to be 'rank:ndcg'.

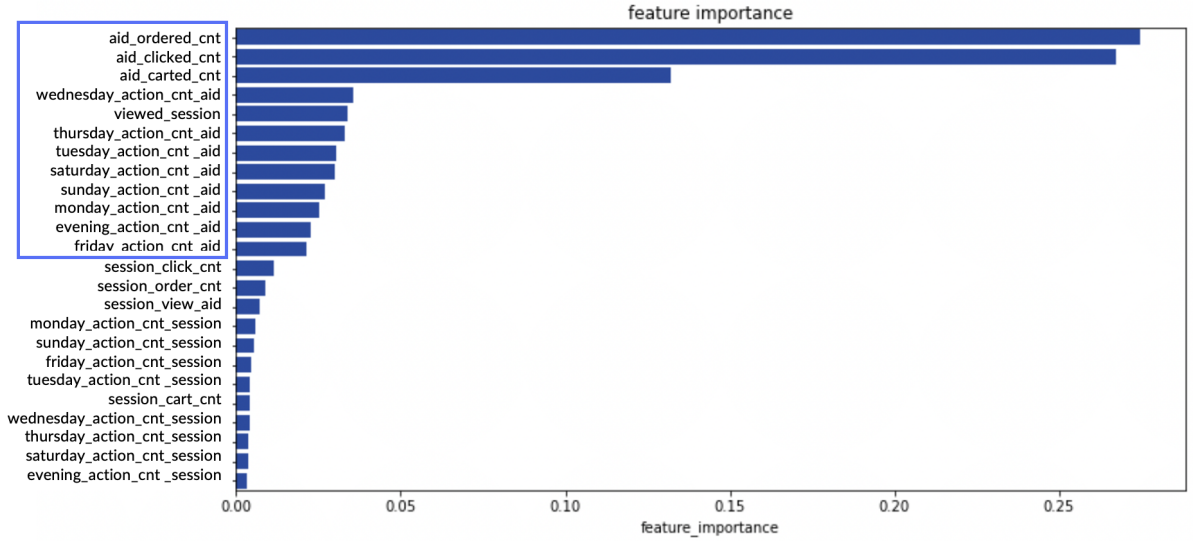
```
from xgboost import XGBRanker

model = XGBRanker(objective='rank:ndcg', n_estimators=100, random_state=0,learning_rate=0.1)
model.fit(
    X_train,
    y_train,
    group=query_list_train,
    eval_metric='ndcg',
    eval_set=[(X_test, y_test)],
    eval_group=[list(query_list_test)],
    verbose =10
)
```

Query group information is required for ranking tasks

One of the things that should be noticed is the parameter 'group'. The training and evaluation data need to be sorted by query group first, and the group must be an array that contains the size of each query group. For example, if the data look like the following, then the input of the group should be an array as [3, 4], which means the first 3 rows belong to the first query and the next 4 rows belong to the second query.

qid	label	features
1	0	x_1
1	1	x_2
1	0	x_3
2	0	x_4
2	1	x_5
2	1	x_6
2	1	x_7

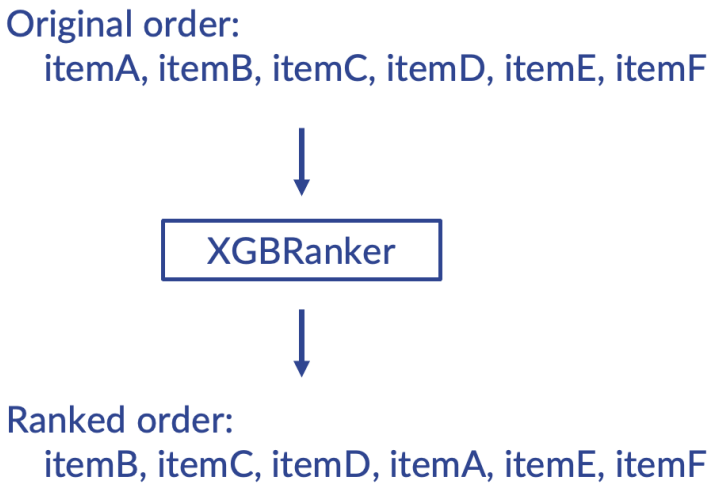
Feature importance:



The feature importance score is calculated by the average gain across all splits that the feature is used in. The top 12 important features happen to be aid related features (features group by aid), which suggests that aid related information is more informative for the ranker model.

Rank items

After training the model, use the model to rank all the items of the sessions in the test set and submit the click, cart, order with the same ranked items.



The score for this submission is 0.462.

Analysis and what we learned from this competition

- Item related features are more important than user related features (in this case). After analyzing the average gain of the features, it's found that the top 12 important features happen to be aid related features.
- The features are not informative enough to capture the patterns of the users. The score of this model is only 0.462, which suggests that the features for the ranker model are not directly related to the objective of this competition. The query and document

relations such as doc product or cosine similarity of items represented by TF-IDF or Word2Vec should be applied.

- Did not generate candidate items for sessions whose number of items <20, only rank the existing items. From the notebooks shared in this competition and the tutorial shared by class TAs, the ideal structure should be like:
  Item corpus -> Candidates generation -> Ranking
  If the candidates can be generated firstly, and rank the candidate items by the ranker model, then the score should be enhanced.

Team work

We overcame many obstacles in this project. The first one is the physical problem, our computer could not execute the program because the data set is too large. After spending a lot of time running notebooks on Kaggle and researching online. We learned the usage of different storage types and tips to improve memory utilization.
Based on what we learned previously and the tips TA taught in class, we try to implement the recommended system by using Tfidf and Word2vec model.
We also spend a lot of time trying many other difficult methods, however many of the results do not reach our expectations. For example, we tried to add weight to a part of test data in the Tfidf model and get a higher submission score by 0.02. If we have a more powerful computer, we could have time to run the whole data and get a higher score.
Overall, we learned a lot during the whole process and have great collaboration in this project. Thanks for the teaching of the professor and TAs !

- Presentation：許博堯、盧奕潔、范佳琦
- Paper Report：俞懿、田展禾