UNSW SYDNEY

COMP9417 Machine Learning Project:

OTTO – Multi-Objective Recommender System

Group Member:

Meng Xiao(z5298034)

Feiyu Qiao(z5324320)

Huizhe Sun(z5375498)

Xinchen Zou(z5378240)

Bowen Zhao(z5446616)

1. Introduction

1.1 Background

With the development of the internet and end-to-end supply chain, online shopping platforms where users can interact, share their preferences, and make purchasing decisions have become incredibly prevalent. As these platforms expand in scale, it has become increasingly challenging to manually monitor and identify user behaviors such as clicks, cart additions, and orders. Thus, researchers have turned to machine learning to help recognize these types of user interactions on a larger scale which could help to improve the shopping experience for everyone involved and provide more personalized suggestions to customers, while online retailers might see an increase in their sales volumes.

In this project, the chance to implement the recommender system is provided by Kaggle and Otto: OTTO – Multi-Objective Recommender System. This task is designed to help online participants select more relevant items from a wide range and make recommendations based on customers' real-time behavior. Improved recommendations will ensure an easier and engaging shopping environment for the customers. The dataset for this project was provided by GmbH & Co KG and can be downloaded from Kaggle platform.

1.2 Aim

Our team aims to construct diverse types of models to distinguish whether the user behaviors are relative to their interests' products. We also try to combine some best implementations from the solutions on the Kaggle platform to try to approach one specific useful model to recommend products.

1.3 Implementation Process

1.3.1 Data Analysis

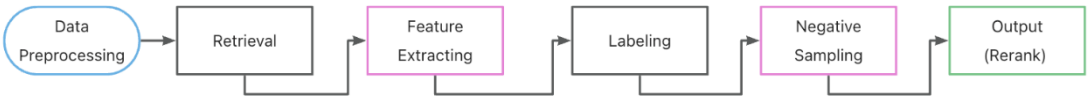
The dataset is combined with a wide range of user behaviors from different users over an extended period. Retrieving the value data for our model and trying to understand which type of dataset should be used in our model is vitally important in this project.

1.3.2 Data Preprocessing

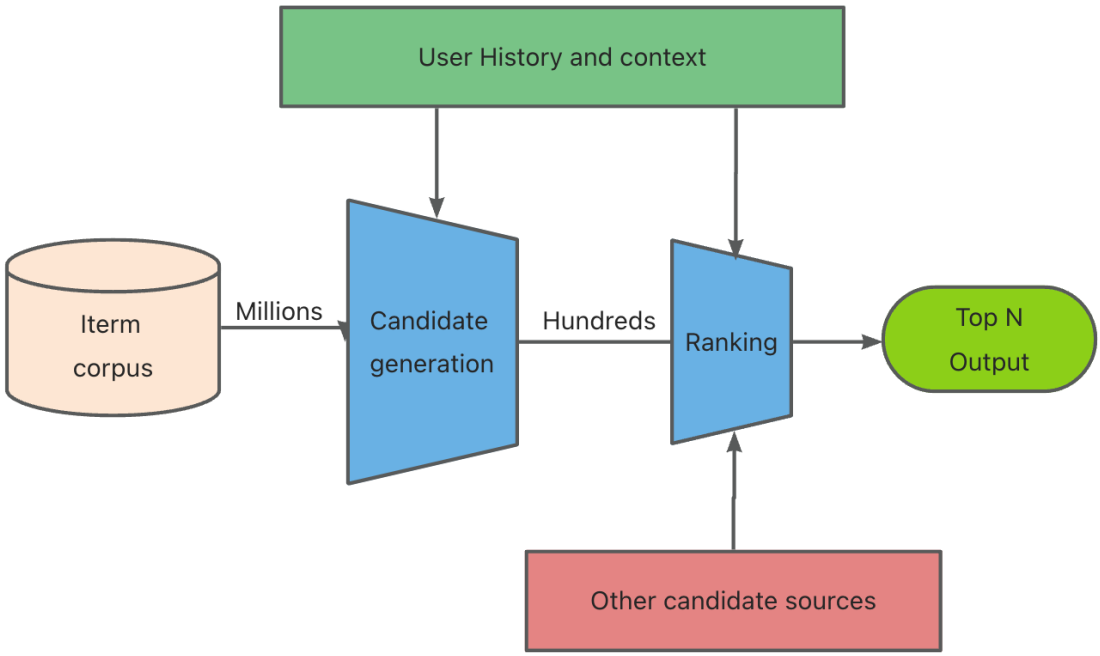
The user behaviors are stored in plenty of different sessions and each session comes from one user, which means that we should combine the behaviors such as click, carts and orders from the user level and then try to train based on such dataset.

1.3.3 Model Design

We implement our model through the following methods: Data preprocessing, data retrieval, feature extraction, labelling, negative sampling and re-rank to generate our final output.



2. Implementation



Basic architecture of the recommendation system

In this part, we delve into the various components factored into the construction of the ultimate model suite. This process consists of the extraction of features from our data set, the assessment of learning algorithms appropriate for our task, the exploration of ensemble algorithms, and the implementation of a model algorithm selection process. Finally, we evaluate our models' performance against with our test data.

2.1 Data Preprocessing

First, we randomly choose 3 million data from the original dataset which consists of 10 million users' information.

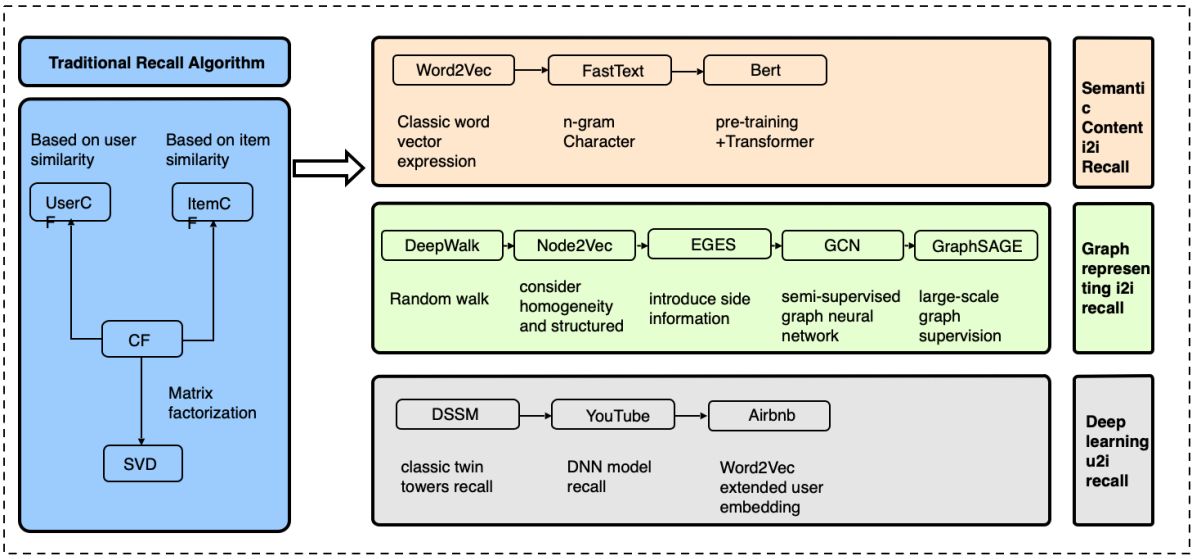
Secondly, to improve the speed of model training, we transfer the large origin date into many small parquet files, which is a columnar storage format that supports sculpture structures and enables them to process more quickly.

2.2 Recall Stage

2.2.1 Implement methods

In the area of machine learning, the Recall Stage is a vital important component of a recommender system or information retrieval system. In this stage, the main goal is to efficiently filter out the relatively small set of user-relevant items from a set of possible candidate items.

The key to this layer is its ability to handle large-scale data. For example, on an e-commerce website, there may be millions of products to choose from. The task of the recall layer is to select a small number of items that may be relevant to the user from these items. This layer typically uses simple, efficient models to provide fast candidate screening. Recall strategies can be based on historical behavior data, item metadata, content-based screening, collaborative filtering, etc. [1]. Recall is the first stage of the recommendation system. It mainly retrieves a small number of items that users are potentially interested in from the massive item library based on the characteristics of users and items, and then hand them over to the sorting link. The amount of data that needs to be processed in this part is very large, and the speed is required to be fast. All the strategies, models and features used should not be too complicated.



The relationship of Recall Model

In our project, we basically use the content semantics to implement our recall stage.

Firstly, we generated the co-visitation matrix, which is mostly in the Kaggle competition to generate the recommendations, it is like the Item Collaborative Filtering (ItemCF) strategy. The method's main advantage lies in its substantial recall rate and low computational cost. The following is the main process of the co-visitation matrix [2]:

1. For each user session, we identify pairs of product events that occur within short time intervals (e.g., less than one day). We then accumulate weights for these product event pairs based on various parameters, such as type, time, and occurrence. This process allows us to establish a co-visitation matrix, represented as M_ {aid1, aid2}, where 'aid' denotes the unique identifier of each product.

2. For each unique product identifier (aid1), we identify the top **'N'** product identifiers (aids) that have the highest accumulated weight. Subsequently, we traverse the list of products that have recently interacted with the target session. Using the co-visitation matrix, we identify the top **'N'** products that are similar. Then, we count the frequency of each similar product. Finally, we select the top **'k'** products with the highest frequency as the recommendations for the target session.

3. The co-visitation matrix utilizes user-defined weights to compute the cumulative occurrence of product pairs, thereby identifying similar items. Suppose the last product in a user session's historical behavior record is X. The top 3 similar products to X in the co-visitation matrix are denoted as A, B, and C, which are termed as the '1-hop' from X. For product A, the top 3 co-visited products are D, E, and F, creating a '2-hop'. After the third hop, we can generate a recall set of 3^3 products for X.

Secondly, we use Word2Vec embedding to compute whether the behaviors could be the relative actions of one user. Word2Vec, while not directly modeling the interest similarity between users and items, does model the similarity between items. By doing so, we can use the items a user has recently interacted with to recall other similar items, allowing the recall results to be updated in real time.[3] Word2Vec, first used in natural language processing, maps discrete words into continuous, low-dimensional vectors. This technique initially applies a sliding window to capture the context of words. In the Skip-gram model, for example, the middle word 'c' is projected into a vector. Then, the probability of any word 's' appearing in 'c's context is:

$$P(w_s|w_c) = \sigma(w_s^T w_c)$$

We randomly sample k distinct words from the dictionary as negative samples, ensuring that these words do not appear in the context of word 'c' to avoid conflicts with positive samples. In this way, we transform the training problem of Skip-gram into a binary classification problem. Log loss can be used as the loss function, and the stochastic gradient descent algorithm can be deployed as the optimizer. After the training is completed, we can obtain the word vectors.

$$J = \sum_{(s,c)\in D+} \log \sigma(w_s^T w_c) + \sum_{(s,c)\in D-} \log \sigma(-w_s^T w_c)$$

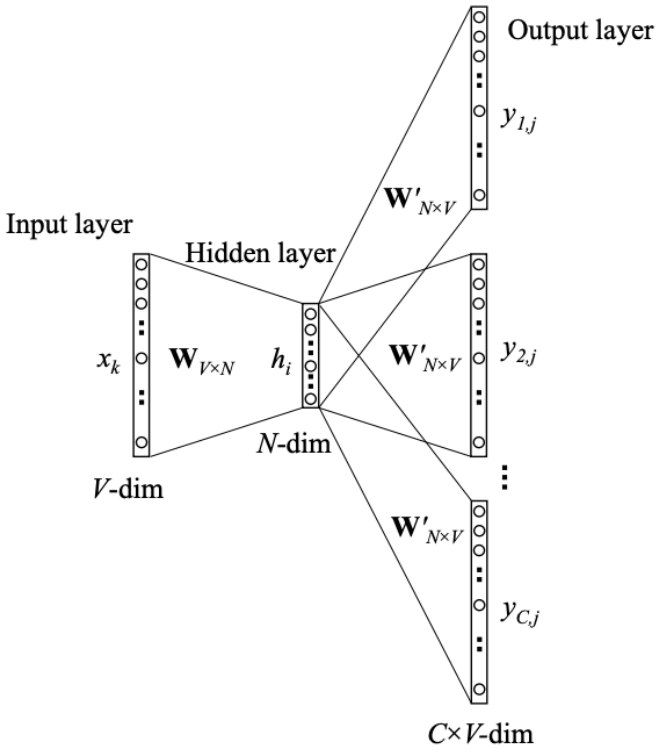


Diagram of Skip-gram model

When word2vec is used as a recall model, the corpus is usually constructed with click sequences. The ad ID sequence clicked by the user in a session is regarded as a "sentence", and the ad ID is regarded as a word. After building the corpus, you can start training. The word vector after training is the embedding vector of item ID, which can be used for recall. When recalling online, we use the item that the user has clicked recently as the query, and sequentially

retrieve N other items closest to each item as the recall result. We can give the historical product events under a user session, represented as {aid1, aid3, aid5, …}, as a sentence to train with word2vec. This training yields word embeddings, which in this case, can be interpreted as product embeddings. Furthermore, we generate user embeddings by averaging or clustering the product embeddings derived from the target user's behavioral history.

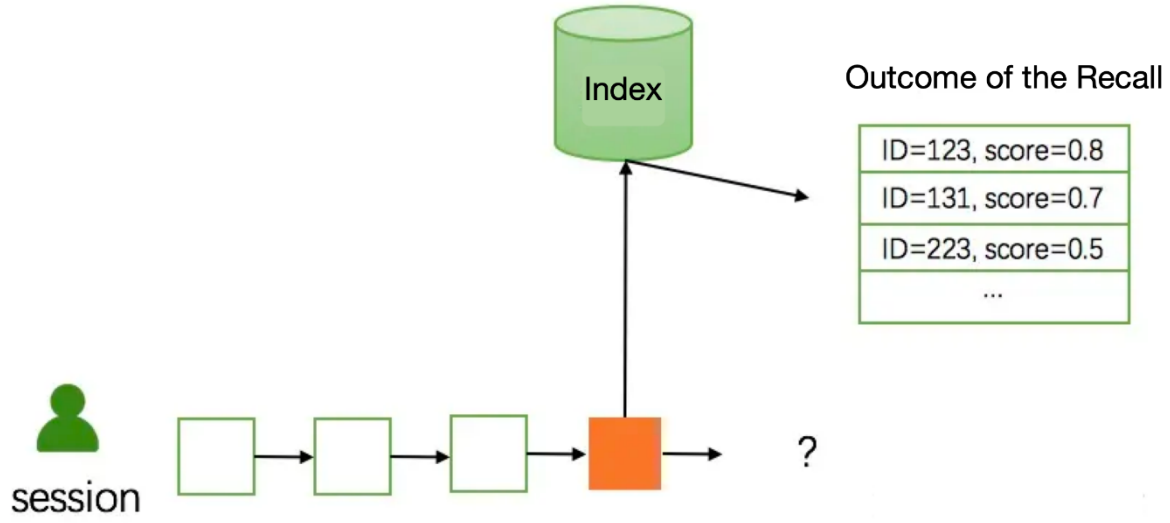


Diagram of item2item recall

2.2.2 Other method discussion

However, in OTTO competitions, the recall strategies that arise. For example, besides the Co-visitation matrix and Word2Vec we mentioned above, we also have Simple Business Policy, Collaborative filtering, and Matrix Factorization.

1. Simple business strategy:

   Based on straightforward business strategy, we can also devise corresponding business recall strategies. For instance, we could gather a collection of products with which the user has historically interacted. Products previously added to the shopping cart are likely to be purchased in the future, and products that have been clicked or purchased repeatedly in the past are also likely to be added to the shopping cart or purchased again in the future. Another strategy could involve collecting popular products, given that users may choose best-selling items during their purchase process.

2. Collaborative filtering:

Collaborative Filtering (CF) leverages user behavior information to filter out the top k products that the target user is likely to be interested in for recommendation. Based on the dimensions used to find similarities, CF can be categorized into UserCF and ItemCF.

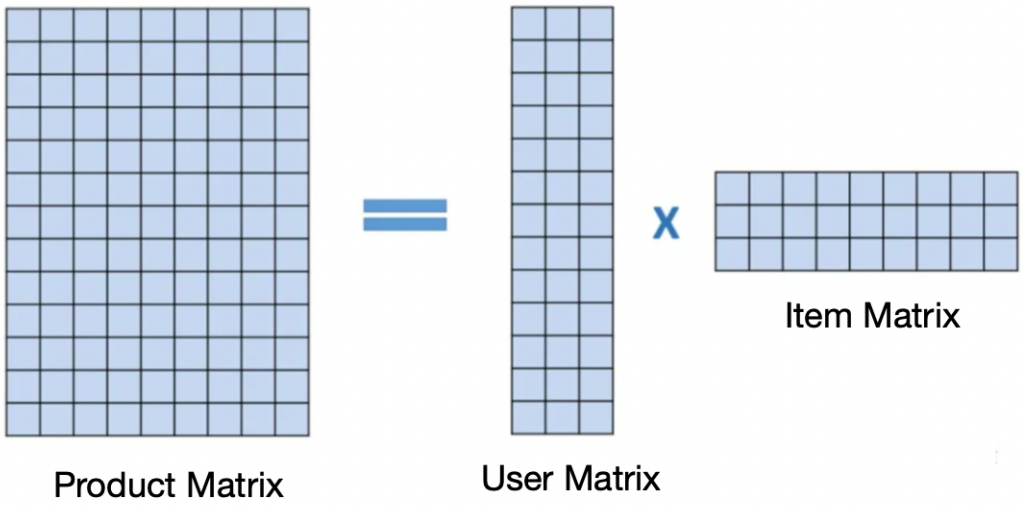
- UserCF finds similar users and predicts the preferences of target users for potential products based on the ratings of similar users on various products. This method first calculates the similarity between user vectors. After finding the top N related products for the target user, it calculates the product score based on user similarity, obtains the product vector score of the target user, and selects the top k products for recommendation after sorting.

- ItemCF identifies equivalent products and recommends products that have high similarity with the target user's historically positively rated products. This method first calculates the similarity between product vectors. Then, for each product 'i' in the target user's historical positive feedback product list, it calculates the product similarity score (product 'i', other potential product 'j') * product 'i's' score. After accumulating weighted scores, it calculates the possible ratings of target users on potential products and finally recommends the top k products after sorting.

The distinction between UserCF and ItemCF lies in the fact that the user count frequently surpasses the number of items. For instance, in OTTO, tens of millions of user sessions are contrasted with only millions of items. Consequently, the storage overhead of the UserCF's user similarity matrix exceeds that of ItemCF. Nevertheless, UserCF exhibits prominent social characteristics, rendering it suitable for contexts with abundant historical user behavior data and rapid interest changes, like in news platforms. Conversely, ItemCF, despite its smaller storage overhead and relatively dense product vectors (excluding some infrequent products), is more apt for environments with stable interest changes, such as video platforms. However, it tends to favor popular products, often neglecting to recommend lesser-known or 'tail' products. [4]

3. Matrix Factorization:

The biggest problem with collaborative filtering is the sparsity of the user-product matrix, so matrix decomposition introduces hidden vectors and decomposes the user-product matrix (m x n) into a user matrix (m x k) and a product matrix (k x n). After solving two small matrices, which can be multiplied together to restore a scoring matrix without missing items, and then recommend items to target users based on the solution scores of all items. [5]



2.3 Feature Extraction

Feature extraction engineering can be roughly divided into the following four categories, the most important ones are recall features, then session & item interaction features, and finally session and item features.

1. Recall features: The features from the recall strategy are most helpful for ranking.
    a. We consider the rank and score of candidate products under each recall strategy.
    b. We need to take into account the number of occurrences of the candidate item in the multi-way recall strategy.
    c. In the item2item CF features, we include the number of occurrences, time difference, sequence distance, and others.
    d. We incorporate BPR features, which involve the vectors of sessions and commodities.
2. Session features:
    a. Statistical characteristics of the time and type of the session's initial and most recent events are considered.
    b. An analysis is conducted on the statistical characteristics of the difference between the session's last occurrence and the last occurrence of all sessions.

c. The study examines the statistical characteristics related to the quantity of different types and the quantity of products in the session.

d. The review focuses on the statistical characteristics of click-to-cart rate, cart-to-order rate, and click-to-order rate within the session.

3. Item features:

    a. Under different time windows, product popularity and type ratio.

    b. Time of the first/last event of the item.

    c. Statistical characteristics of click-to-cart rate, cart-to-order rate, click-to-order rate in the same session.

4. Session & Item features: Make interaction features between sessions and items.

    a. Determine the index position at which the candidate from a session appears in the historical interactive product list.

    b. Calculate the number of sessions or the proportion of total sessions where the candidate item is present in the historical interaction list.

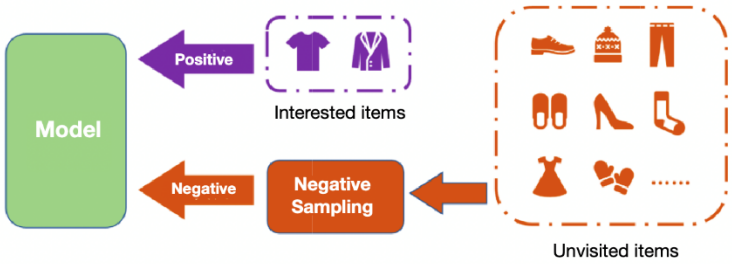
    c. Examine the time and type of the first and last event of a session.

2.4 Negative Sampling

In the context of recommendations, the recommendation model primarily uses a user's historical feedback to model their interests. During model training, both positive examples (products the user likes) and negative examples (products the user doesn’t like) are typically provided to the model. The model then learns representations of users and products based on the loss function, ultimately facilitating model training. However, obtaining explicit feedback from users (such as product ratings) can be challenging in real-world recommendation scenarios, making it difficult to definitively ascertain which products users prefer and which they don't.

Mostly, the available data consists of user's implicit feedback, such as the record of products consumed by the user. This implicit feedback does not have clear labels. For training the model, it's generally assumed that all products a user interacts with are positive examples, while a subset of products the user has not interacted with are selected as negative examples through

sampling. This process of selecting negative examples from the pool of products not interacted with by the user, based on a certain strategy, is referred to as negative sampling. [6]



In our project, considering the size of the positive sample and the scoring weight for each type, the ratios of positive to negative samples under 'click', 'cart', and 'order' can be identical, or the 'order' negative sample ratio may exceed the 'click/cart' negative sample ratios. For instance, the ratios can be 5% for clicks, 25% for carts, and 40% for orders. Negative sampling serves to reduce the volume of data input to the subsequent ranking model. Although this approach may sacrifice some accuracy, it accelerates training iterations. [6]

3. Experiments and Results

3.1 Experiment setting

In our experiment, the code we developed needed to run in the Kaggle environment. Kaggle provides a cloud-based platform where we can execute our code and access various datasets. To run the code in the Kaggle environment, we uploaded the code to the Kaggle workspace and configured the necessary settings. The specific implementation details and code snippets can be found in the code repository or supplementary materials provided.

To enhance the speed of predicting for each session-type, we utilized parallel CPU processing with the list data format. Parallel computing is a technique that involves executing multiple tasks simultaneously to improve computational speed. In the Kaggle environment, we can leverage the power of multiple CPU cores by specifying the number of CPUs to be used. In our experiment, we utilized four CPU cores for predicating clicks, carts and orders for each session.

3.2 Evaluation metrics

Recall@20 scores of the e-commerce clicks, cart additions, and orders are used to evaluate the predictions. The final scoring formula that encompasses the three Recall scores and the formula for calculating Recall@20 are as shown below. The first formula highlights that the Recall of order-related sessions carries the highest weight, emphasizing the importance of accurately identifying actual orders. In the second formula, N denotes the total number of the sessions of a session-type (clicks, carts or orders), predicted aids refers to the first 20 predictions for a session-type, and ground truth aids correspond to the labeled aids in the test set.

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

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

3.3 Experiment results

The final prediction format is as follows: for each session in the test set, the predicted aid values are provided in the label column, which is space delimited, and in each row, up to 20 aid values are presented. An example of the format of predication is as shown below.

	session_type	labels
0	12899779_clicks	59625 1790770 637538 941596 1246235 273918 448...
1	12899780_clicks	1142000 736515 973453 582732 889686 636101 487...
2	12899781_clicks	918667 199008 194067 57315 141736 1460571 9507...
3	12899782_clicks	834354 595994 740494 889671 987399 779477 1344...
4	12899783_clicks	1817895 607638 1754419 1216820 1729553 300127 ...

After evaluation on the test set of the published OTTO Recommender Systems Dataset, our method achieved a private score of 0.5645 and a public score of 0.56417. The results of our experiments indicate that our implemented recommender system achieved satisfactory performance. By combining various techniques including the co-visitation matrix generation and the Word2Vec embedding, we were able to effectively filter out relevant items for users. The recall rate was significantly improved, leading to more accurate and personalized recommendations.

The use of content semantics in the recall stage allowed us to capture the similarity between items and leverage user behaviours to improve the recommendation process. This approach proved to be efficient and computationally cost-effective. However, there is still room for improvement in our model. Further exploration of ensemble algorithms and the incorporation of additional features could potentially enhance the recommendation accuracy. Additionally, fine-tuning the parameters of our model and experimenting with different weighting schemes may lead to better performance.

In terms of computational efficiency, our model showed acceptable performance. The running time for predicting the clicks, carts and orders for 1,671,803 sessions in the test set are as shown in the table below. From the table, we can find that our system is able to predict for over 1 million sessions within two minutes, indicating that our method achieved improved runtime and efficiency in predicting the results by leveraging parallel CPU processing and utilizing the list data format.

Table 1. Running time for predicting the clicks, carts and orders

Session-type	*CPU time (user)*	*CPU time (system)*	*CPU time (total)*	*Wall time*
clicks	29.8 s	2.95 s	32.7 s	53.9 s
carts and orders	32.5 s	3.45 s	35.9 s	1min 5s

Our implemented recommender system demonstrated promising results in the OTTO Kaggle competition. By effectively filtering out relevant items and providing personalized recommendations, we will be able to create a more engaging shopping environment for customers.

4. Conclusions, learnings, and further work

4.1 Conclusions

In this project, our aim was to implement a recommender system using the OTTO Recommender Systems Dataset provided in Kaggle competition. The purpose of this task was to improve the relevance of item recommendations based on real-time customer behavior, thereby creating a more engaging shopping experience. To achieve this, we constructed various models to determine the relationship between user behaviors and their interests in products. We also leveraged the best implementations from the Kaggle platform to develop a specific and useful recommendation model. We focused on the several components involved in building our ultimate system. This included the selection and implementation of appropriate learning algorithms, feature extraction (especially for recall features, the session & item interaction features, and finally session and item features), and negative sampling. In the recall stage, the main goal was to efficiently filter out user-relevant items from a set of candidate items. We achieved this by using content semantics, specifically the co-visitation matrix and Word2Vec embedding. These methods allowed us to generate recommendations based on item similarity and user behaviors.

4.1.1 Word2vec

This method calls the existing library directly and adjusts the parameters to implement the recommendation function. The specific logic is to select the latest 20 products to recommend through the generated dot plot. But when the number is less than 20, fill these vacant positions with more popular goods. And in the middle, we define a function called hashfxn that initializes the output to 0 and stores the cumulative hash value. The optimization in this way can accelerate the access and matching of data in some cases and reduce the occupied memory because the data is more compact, which is more suitable for the use scenario of processing a large amount of data in our project. And it did get better results. After using the word2vec model for text processing and adjusting hyperparameters, our training accuracy reached 49%. Through observation, we feel that there are no overfitting problems during the whole training process, which can verify the validity of our hyperparameters to some extent.

4.1.2 Co-visitation matrix:

After initializing the partition of the disk, this part of the code iterates through the processing of the data document and establishes a user behavior network according to the conditions and time stamps and recommends products to users by calculating the degree of correlation between these behaviors. Similarly, the prediction speed was very slow at the beginning, and then the parallel CPU and the method of converting parallel frame into list were used, which greatly shortened the operation time and improved the performance efficiency of the model.

We evaluated the performance of our model on the provided test data. The evaluation of our models was based on the Recall@20 metric for each action type (clicks, cart additions, and orders), with weights assigned to each type. The final private and public scores were 0.5645 and 0.56417, respectively. In terms of computational efficiency, the click prediction task took approximately 53.9 seconds, while the cart-order prediction task took around 1 minute and 5 seconds. In conclusion, our project successfully implemented a recommender system using the OTTO Recommender Systems Dataset. The results showed improved recommendations and a more engaging shopping environment for customers. Further improvements such as exploration of ensemble algorithms, the incorporation of additional feature, and fine-tuning, can be made to enhance the accuracy and efficiency of our system.

4.2 Further work

In the process of data set processing, we currently only adjust the size of the data set to match our own project. But a later look at the data found that false labels are a good way to improve the model when there are discrepancies between the training and test datasets. In the future, we can use this direction to get a more balanced and reasonable data set, and then train it. In this way, we may be able to get more accurate results on models that are known to be used.

We plan to solve the problem by combining two different composite models, specifically word2vec+co-visitation matrix, as each model will have its limitations. But at the same time, our concern is that the prediction is not always accurate, so after the inaccurate data is merged with the model, the result may be even more inaccurate. This is a direction that we may study and solve in the future.

Finally, Word2Vec and co-visitation matrix are both neural network-based data analysis methods used to find potential relationships, themes, correlations, etc., between items. However, we can also try NN model, a neural network model, which may be more in line with the application scenario of this project and have more outstanding flexibility and performance ability.

5. Reference

[1] https://arxiv.org/pdf/1411.2738.pdf

[2] https://www.kaggle.com/code/vslaykovsky/co-visitation-matrix/notebook

[3] https://arxiv.org/pdf/1310.4546.pdf

[4] https://github.com/datawhalechina/funrec/blob/master/docs/ch02/ch2.1/ch2.1.1/itemcf.md

[5] https://ieeexplore.ieee.org/document/5197422

[6] http://wnzhang.net/papers/lambdarankcf-sigir.pdf

[7] https://www.kaggle.com/competitions/otto-recommender-system/discussion/384022