DSA5201 - Final Report
Recommendation System Development and Optimization

Liu Boyu (A0177847J)

Abstract
In this report, I will present what I have done and what I have learnt during my work in Shopee as a full time employee. The main focus of this report will be on the projects I am involved in, namely flash sale search and product collections auto generation, and describe the difficulties I encountered in each project, as well as the methodologies I applied to solve the problem. Meanwhile, I will also talk about how the knowledge and skills gained in NUS helped me with my work in Shopee.

Table of Contents

Introduction

During the period of taking DSA5201, I work at Shopee as a full time employee and my role is algorithm engineer. In my daily work, my job focuses on recommendation systems and search algorithms.

Recommendation systems are a very popular and effective paradigm in retail business. With a recommendation system, shoppers can find items they like with less effort. Furthermore, they are presented with items they've never thought of buying, but which actually suits their needs.

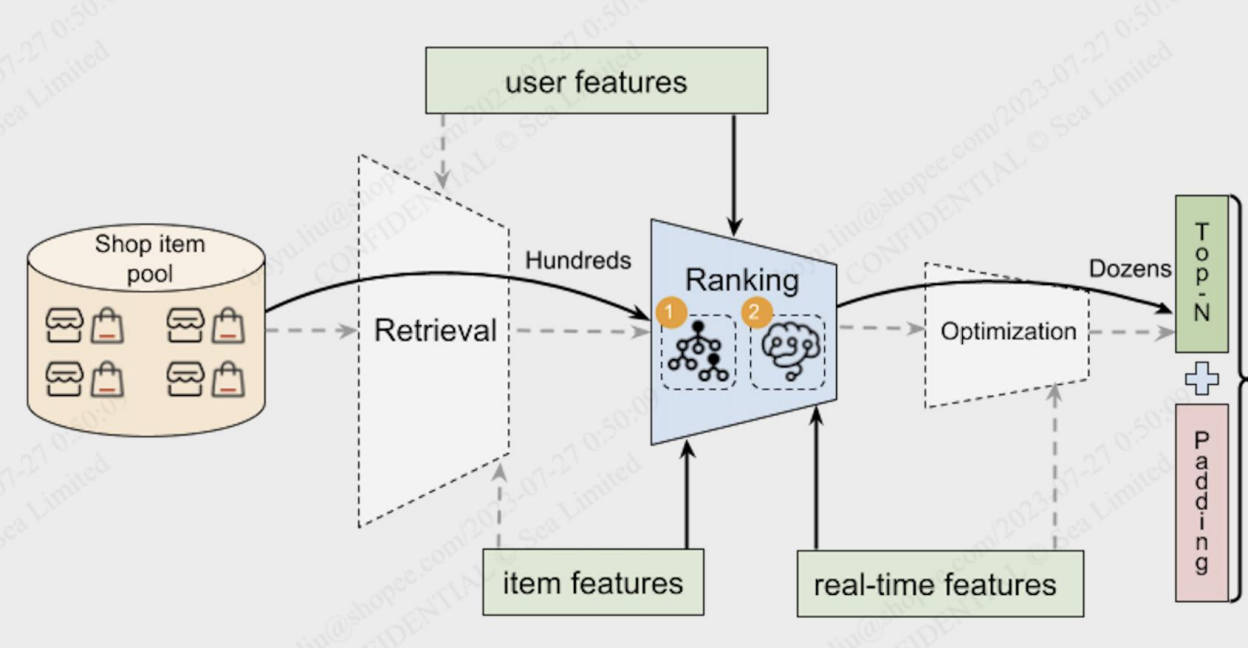

Figure 1: Example of Standard Recommendation System Structure

Figure 1 shows an example of the standard recommendation system that we are using currently in Shopee. It starts with a large item pool, usually containing millions of items. A retrieval system will retrieve thousands of items from the item pool, and send them into the ranker. The ranker will rank each item by user-features and item-features, and assign a score to each item. After that, an optional re-ranker is applied to further optimize the ranking order. Finally, the sorted item list based on their order from ranker will be returned to frontend and displayed to users.

Search algorithms, on the other hand, have a similar structure as the recommendation system, but have a different focus. In our system design, search algorithms share the same workflow with recommendation system as indicated in figure 1, but the retrieval system and ranker are completely different as search algorithms focus more on the relevance between user's key-in query and items, while recommendation systems only care about how interested user is in each item.

My work in Shopee includes development and optimization of all the components in a recommendation system. But in the past three months, my main focus are on item pool,

generation, retrieval system design and re-ranker implementation. And I will talk about these three components one by one through my project report regarding flash sale search project and product collection auto generation project.

Data

In the development of these projects, multiple sources of data are used and there is no predefined and fixed schema for the data as different business scenarios have different schema for their own special needs. **All data are extracted from Shopee raw data logged by frontend trackers respect to the needs of the projects**. In fact, one of the main work I have done during taking DSA5201 is to design the schema of the data log, and I will talk about it in the later part.

```
root
 |-- keyword: string (nullable = true)
 |-- keyword_cluster: string (nullable = true)
 |-- level1_keyword_category: string (nullable = true)
 |-- level2_keyword_category: string (nullable = true)
 |-- keyword_search_volume: long (nullable = true)
 |-- keyword_search_volume_rank: long (nullable = true)
 |-- keyword_cumulative_search_volume_percentile: double (nullable = true)
 |-- keyword_search_volume_type: string (nullable = true)
 |-- local_date: date (nullable = true)
 |-- region: string (nullable = true)
```

Figure 2: Sample Schema of one of the data source used in projects

Flash Sale Search

Flash Sale Search is one of the highest priority projects in our team. It is made up by two components: Flash Sale, and Search.
Flash sale is a project that our team have been working on for more than 2 years. It sells on discount product, and is guaranteed to have lowest price within platform. It is a highly mature project and it's one of the most important projects in the whole company. The figure below shows an example of flash sale on the shopee website.

Figure 3: Flash Sale on Shopee

Just as the name indicated, we would like to introduce a search feature in our existing flash sale project so that users can directly search for their wanted items to see whether it is in a promotion.

Flash Sale Search - Infrastructure Implementation

As our team only focuses on recommendation systems before, and none of our team members have done search algorithm related projects before, there is no usable infrastructure that I can use for the Flash Sale Search project. The Infrastructure here means formatted data can be used as features to develop models, and text understanding models such as BERT. Therefore, I will need to prepare formatted data from raw log of events through pyspark, a distributed processing system used for big data processing, to be used as feature for further model training.

```
root
 |-- keyword: string (nullable = true)
 |-- keyword_id: long (nullable = true)
 |-- click_item_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- click_item_l1_cat_id_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- click_item_l2_cat_id_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- click_item_l3_cat_id_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- click_item_timestamp_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- add_to_cart_item_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- add_to_cart_item_l1_cat_id_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- add_to_cart_item_l2_cat_id_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- add_to_cart_item_l3_cat_id_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- add_to_cart_item_timestamp_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- order_item_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- order_item_l1_cat_id_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- order_item_l2_cat_id_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- order_item_l3_cat_id_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- order_item_timestamp_seq: array (nullable = true)
 |    |-- element: long (containsNull = true)
 |-- grass_region: string (nullable = true)
 |-- grass_date: date (nullable = true)
```

Figure 4: Data schema for one of feature prepared for model training

Flash Sale Search - Recall Logic Design

As Flash Sale Search is a newly initiated project, the recall logic needs to be re-designed based on business scenario requirements. In previous projects, there is only one item pool as shown in Figure 1, but in this projects, we have 4 item pools and we need to retrieve items from all of the simultaneously. One of the problems this brings is how we design the merge logic between different item pools as the score for items from different pools may not be comparable. Besides, as requested by the PM side, these four item pools should have different priorities. To meet all the requirement and guarantee a smooth merge process, I designed a logic to reassign score to

items retrieved from different pools to make them comparable, and assign a weight to different item pools to make sure they have different priorities.

```
for each item list l, pool score weight w pair:
    sort(l) by relevance score in descending order
    if len(l) == 0:
        result_list = []
    elif len(l) == 1:
        result_list = [maximum * w]
    else:
        result_list = []
        for each index, item in l:
            normalised_weighted_score = (maximum - index * ((maximum - minimum) / (len(l) - 1)) * w
            result_list.append(normalised_weighted_score)
    return result_list
```

Figure 5: Pseudo Code for Recall Logic

In the above pseudo code, the original score returned by retrieval system to compare items is not used, but to assign a new score to each item based on the position.
For each of the four pools, a retrieval is done, and generates in total four item lists. After that, a new score is assigned to each item in eac list based on their position, so that scores in all liss become comparable.
Eventually, a different weight is assigned to different list, and merge them by the final weighted score. And the rest shall just follow standard workflows.

Flash Sale Search - Deep Listwise Context Model

As the recall logic and features ready, next I will need to build re-ranker for click-through rate and conversion rate improvement. Deep Listwise Context Model (DLCM), as the name indicated, is a listwise ranker. Compare to traditional point wise ranker, it directly look at the entire list of items and try to come up with the optimal ordering for it.

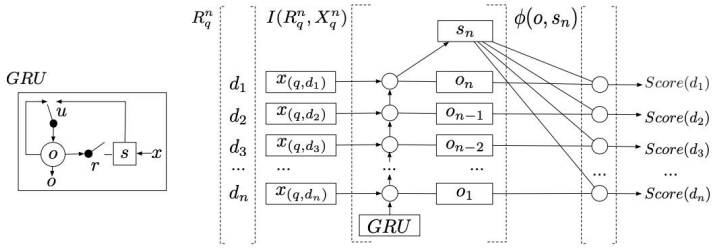

Figure 1: The overall structure of the Deep Listwise Context Model (DLCM). R_q^n is a ranked list provided by a global ranking function f for query q; $x_{(q,d_i)}$ is the feature vector for document d_i; s_n and o_i is the final network state and hidden outputs of the RNN with GRU in $I(R_q^n, X_q^n)$; and $Score(d_i)$ is the final ranking score of d_i computed with $\phi(o_{n+1-i}, s_n)$

Figure 6: Model Structure of DLCM

The above figure shows the structure of DLCM. It takes in a list of items sorted by previous ranker (as it is a re-ranker) to go through a RNN layer, and generate a sequence of outputs and a final state, which is a vector representing the encoded item list. After that, the following

$$\phi(\boldsymbol{o_{n+1-i}}, \boldsymbol{s_n}) = \boldsymbol{V_\phi} \cdot \left(\boldsymbol{o_{n+1-i}} \cdot \tanh(\boldsymbol{W_\phi} \cdot \boldsymbol{s_n} + \boldsymbol{b_\phi})\right)$$

Equation 1: From sequence outputs to final ranking score

Equation is applied to calculate the predicted ranking score for each item in item list.

With ranking score calculated, next is to calculate the loss. As DLCM tries to optimize the order for the whole list instead of each item, it uses an attention loss to do so.

$$a_i^y = \frac{\psi(y_{(q,d_i)})}{\sum_{d_k \in R_q^n} \psi(y_{(q,d_k)})}$$

Equation 2: Attention Allocation Strategy

DLCM first computes true attention label by applying the attention allocation strategy in equation 2, and then compute attention score for the DLCM ranked list. After that, a standard cross-entropy loss was calculated as final loss of the model.

Product Collection - Auto Generation

The objective of product collection auto generation is to detect what's trending in society, and make an product collection related to that trending. To introduce an overview of this project, imagine that now is graduation season, and students want to buy flowers for their graduation ceremony. Instead of letting them search flowers themselves, we can somehow detect the need of graduation flowers, and generate a collection of these kinds of items, and push to users that need it.

To find out what should be made to a collection, the ability of detect trending is needed. There are two types of trend that we want to detect: sudden trend and periodic trend.

Product Collection - Auto Generation - Sudden Trend

The algorithm for detecting sudden trend is based on twitter's trending detection algorithm. It's basic assumption is that, search volume of a hash tag should follow poisson distribution. In poisson, many sources have individually low probabilities of contributing to the count, which is exactly the case in our observation for items. We also did some changes on our own, like added alick and order data into consideration, so that it matches more with our requirement.

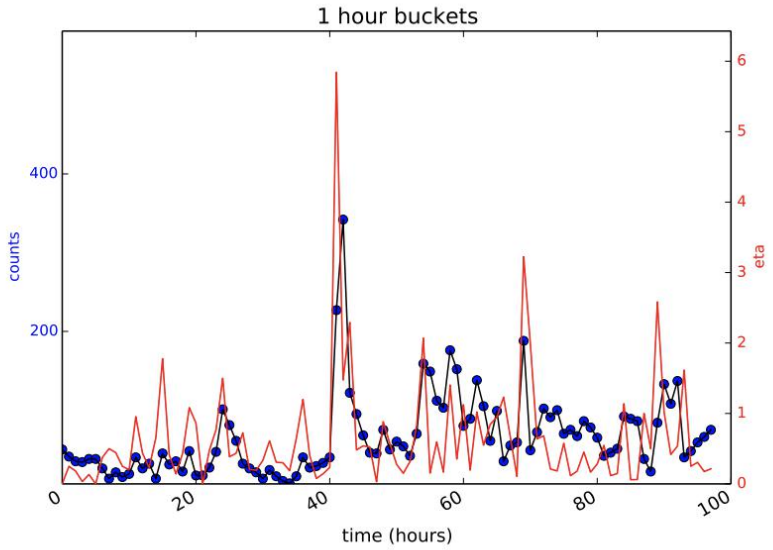

Figure 7: Sample output of sudden trend algorithm

The above figure shows a sample result of the sudden trend algorithm, where blue points is the search counts and red lines are predicted trending probability.

Product Collection - Auto Generation - Periodic Trend

For seasonal trend detection, prophet model published by Facebook was used. It divides data into different components, and hence provide insights on trendings. The figure below shows an example of prophet model output. The left component is the original data, and the four components on the right are basic trend, holiday trend, weekday trend and yearly trend. For sunscreen, the yearly trend goes up in summer times and goes down in winter times, which matches with common sense.

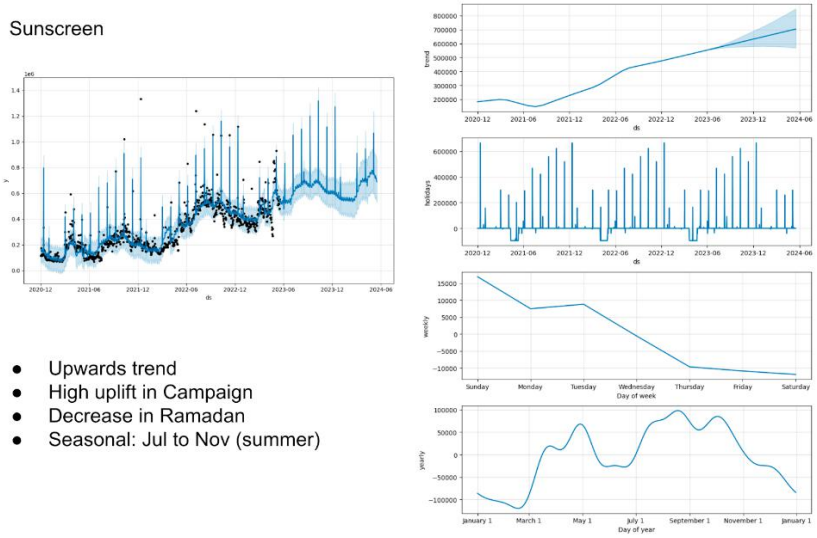

Figure 8: Sample output of periodic trend model

Summary

In summary, during my work in shopee, i am involved in multiple projects like flash sale search and product collection auto generation and i think i have done well.

In my work, not only the academic skills i learnt from NUS was used, like building deep models, analyze datas, but also many soft skills like communication between different team, design logic for engineering team’s need and so on.

I look forward to my continuous work in shopee and hope to make more values.