SHOPEE CODE LEAGUE 2020

GUIDELINES TO THE COMPETITION SOLUTIONS

Updated: 21 Jul 2020

Competition #1: Order Brushing [Data Analytics]

Task:

1. Identify all shops that are deemed to have conducted order brushing.
2. For each shop that is identified to have conducted order brushing, identify the buyers suspected to have conducted order brushing for that shop.

For the purpose of this question, shops are deemed to have conducted order brushing if their concentrate rate is greater than or equal to 3 at any instance

```
Concentrate rate = Number of Orders within 1 hour / Number of Unique Buyers within 1 hour
```

For the purpose of this question, suspicious buyers are deemed as the buyer that contributed the highest proportion of orders to a shop that is deemed to have conducted order brushing.

For calculation of the highest proportion of orders to a shop, only include the orders that occured in instances when order brushing has been deemed to have taken place.

In the case where multiple users share the same highest proportion of orders for a specific shop, all those users are deemed to be suspicious buyers.

Approach

The basic idea is calculating Concentrate rate for all possible 1 hour windows. Here illustrate the strategy:

a. Select a 1 hour window and take a slice from order data.

   Note when you set the time condition, both ends are included in the 1 hour window. (Hint: For Python, datetime module and timedelta function can be applied)

b. Find shops which satisfy CR>=3.

   For the slice of order data you select, you can use an aggregation function like groupby to calculate order number and unique user number for each shop in this 1 hour window. And then filter shops which satisfy the CR>=3 condition. They are brushing shops.

c. Find the userid who conducts order brushing in brushing shops.

   When brushing shops are identified, aggregation function can be applied again to calculate which userid contribute the most orders for each shop. Note if two userid both have the highest order number, then they are both the answer.

d. Repeat the above procedure for all possible 1 hour window.

   For example: 2019-12-27 00:00:00 ~ 2019-12-27 01:00:00, 2019-12-27 00:00:01 ~ 2019-12-27 01:00:01, …

Note:

When you follow the above steps, you may find **d** step takes a long time since the time period in our data crosses 5 days and the time window is detailed to second. To reduce the time consumption, some advanced strategy might be applied. For example:

a. Multiprocessing to brutally calculate them all.
b. Avoid redundant calculation by detecting windows which cover exactly the same orders.

It would be our pleasure if you find your own way to solve it. Build your code and have fun~

Competition #2: Product Detection [Data Science]

Task:
In this competition, a multiple image classification model needs to be built. There are ~100k images within 42 different categories, including essential medical tools like masks, protective suits and thermometers, home & living products like air-conditioner and fashion products like T-shirts, rings, etc. For the data security purpose the category names will be desensitized. The evaluation metrics is top-1 accuracy.

Approach

Preprocessing:

Clean the dataset by some cluster techniques such as p-hash, image histogram and metric learning. Since we know that the most of images within one category are correct, can use such techniques to remove those outliers as much as possible.

For some categories like class No.33 with only ~500 samples, may have less samples than others, you might need to use some augmentation to upsample those categories. Balanced dataset is extremely important for classification, it is always recommended to make dataset balance as possible as you can.

The preprocessing and training strategy may count more than the model itself sometimes.

Data Augmentation:

Some useful techniques for image augmentation are color jittering, jpeg compression, cutout, mixup, label smoothing etc. Those techniques will help to make the model more robust which will let your score more stable for public leaderboard and private leaderboard.

Classification Model:

There are quite a few models that are suitable for this task. ResNet, ResNext, EfficientNet, ResNeSt, VGG, InceptionNet and etc. For such a 40-category classification task, it is recommended to use a relatively larger model to avoid overfitting and also for better performance.

Training strategy:

This part is another important part, but it totally depends on different conditions like different machines, different preprocessing and different optimizers. Basically, if you have a strong machine which can support you to train with a large batch size, then you might use a relatively large learning rate and less epochs. But if your machine cannot support a large batch size training, to get the same level result, you might need to use a smaller learning rate and more epochs. Or you can also use the loss accumulation strategy to simulate larger batch size.

And do note that the training strategy should be decided based on your analysis for validation/public leaderboard result. With some iterations of submissions and analysis, the strategy will be polished to a perfect state.

Competition #3: Shopee Programming Contest #1 [Engineering]

Task: Please refer to the problem statements here.
You may refer to the solution/guideline for each problem statement below.
3.1 Item Stock
3.2 Search Engine
3.3 Judging Servers
3.4 Lucky Winner
3.5 Sequences

3.1 Item Stock

We will calculate the final stock for each fixed stock item first since dynamic stock items will always be derived from them.

For each item, store its lowest ancestor which is a fixed stock and the multiplication of the **Qty** to that ancestor. Then, each time we create a fixed stock item we can easily do the stock deduction in **O(1)**. After we finished calculating the fixed stock items, we can just derive the dynamic stock items. The total complexity is **O(N)**.

3.2 Search Engine

This problem is quite straightforward and requires **Trie Tree** style data structure. This problem can also be solved using other techniques. For each input item name, first, hash the words in the name to an integer. For example, “iPhone SE 2” can be converted to a vector of integer [1 2 3] and in the same way, “iPhone 11 Pro” can be converted to [1 4 5].
Then you need to insert all the suffixes of the vector containing the hash values, into the trie tree data structure.

Each node of the Trie tree will store the number of times this node as part of a different item. One tricky part is to eliminate the overcounts. Consider the number of items in the

input is N and each work can have m space. The overall memory complexity is **O(N * m^2)**. And Time Complexity is **O(Q*m)**.

3.3 Judging Servers

This is a classical dynamic programming problem. Only one tricky part of the problem is that you can choose not to take a free server, actually taking a free server is optional. Any point of time you can be in three different types of state and you have to minimize the cost

// 0 => no back
// 1 => took previous one with free now need to buy the current
// 2 => the next one is free or you can buy.

From state 1,
You took the previous server for free, so you pay for the current one -->? move to state 0.

From state 2,
you can take the current server for free → move to state 0
you can buy it → moves to state 2.
You can skip this server → move to state 0

From State 0,
Skip the current server → move to state 0
Take the current server for free → move to state 1
Buy the current server → move to state 2

Time Complexity and memory, O(N*S) ~ O(N^2)

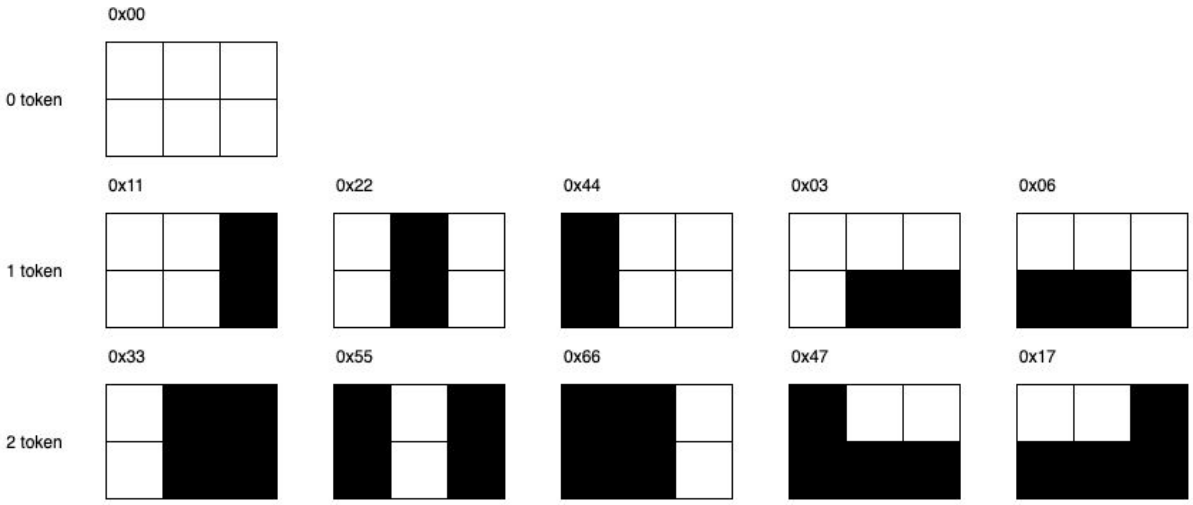
3.4 Lucky Winner

This is a classic dynamic programming problem.

Let DP[i][j][mask] be the maximum worth of items consider that we've used **j tokens** on first **i rows** with **mask** being a bitmask describes the state of **i-th row** (3 bit bitmask, so 8 states in total)

We'll need to calculate this table. Result will be max of DP[n][k][mask] with 0<=mask<8

The solution for a state can be calculated based on the states of the previous row. There are 11 ways to use the tokens in the last 2 rows. Each of these ways needs some available fields in the last row and will take some fields in the current row. To simplify the implementation, we represent each way of placing tokens in the last rows as a bitmask where the first digit is the state of occupied fields in the previous row and the second is the state of occupied fields in the current row. Please refer to the image below about states and their costs.



The number of states is N*K*8 = 8e6, there are 11 ways of using tokens in the last 2 rows. Overall complexity O(N*K)

3.5 Sequences

Let's denote **count(x, y)** as the number of sequences with length **x** which sum modulo **K** equals to **y**. Clearly, the answer we seek will be **count(M, 0)**. Now, observe that **count(a+b, c)** is equal to the summation of **count(a, i)** x **count(b, (c-i))** for each **0 ≤ i < K**. Following this observation, we can calculate the value of **count(x, *)** in similar fashion to exponentiation by squaring. This part can be done in **$O(K^2 \log M)$**, where **K^2** comes from calculating **count(a+b, *)** for any given **a** and **b**.

Now, we need to calculate the base case, which would be **count(1, *)**. Note that this is equal to finding the distribution of elements of each **f(x, y)** from the description in terms of modulo **K**. Observe that for any given sequence, it will be cycled every **K** terms since **$K^2 \bmod K = 0$**. Hence, using this we can calculate **count(1, *)** in **O(K)** for each given function, which totals to **O(NK)** for counting them all.

Combining those, we got a solution with total complexity **$O(K^2 \log M + NK)$.**

Competition #5: Logistics [Data Analytics]

Due to the recent COVID-19 pandemic across the globe, many individuals are increasingly turning to online platforms like Shopee to purchase their daily necessities. This surge in online orders has placed a strain onto Shopee and our logistics providers but customer expectations on the timely delivery of their goods remain high. On-time delivery is arguably one of the most important factors of success in the eCommerce industry and now more than ever, we need to ensure the orders reach our buyers on time in order to build our users' confidence in us.

In order to handle the millions of parcels that need to be delivered everyday, we have engaged multiple logistics providers across the region. Only the best logistics providers that are able to meet Shopee's delivery standards are partnered with us.

The performance of these providers is monitored regularly and each provider is held accountable based on the Service Level Agreements (SLA). Late deliveries are flagged out and penalties are imposed on the providers to ensure they perform their utmost.
The consistent monitoring and process of holding our logistics providers accountable allows us to maintain our promise of timely deliveries to our buyers.

Task:
Identify all the orders that are considered late depending on the Service Level Agreements (SLA) with our Logistics Provider.

For the purpose of this question, assume that all deliveries are considered successful by the second attempt.

Approach
Here are the main difficulties in this problem statement:

1. Processing of large datasets
2. Handling of epoch time
3. Parsing the address fields to extract the relevant states when calculating SLAs
4. Weekend & Public Holiday exemptions when calculating SLAs
5. 1st attempt deliveries & 2nd attempt deliveries

Perhaps the largest challenge would be the exemptions for Weekend & Public Holiday and applying it to both 1st & 2nd attempts

General approach to solving this:

1) Convert all the epoch time format into something that is easy to read and compute.

   For python, this can be done easily with "*pd.to_datetime".* However, do be careful as this will convert the time to GMT timezone. We can change it to GMT+8 by simply adding 8 hours in seconds to the epoch time before applying "*pd.to_datetime"*

2) Extract all the names of the states from both buyer and seller addresses.

   A simple string match or regex should yield the correct states

3) Get # of days between pickup and 1st delivery attempt

   We ignore the time and consider only the date. Since we have previously converted our epoch time using "*pd.to_datetime*", we can continue using Pandas' datetime functions. We can get the difference between the 2 datetimes and apply "dt.days" to get the number of days. Do the same for 1st delivery attempts and 2nd delivery attempts.

4) Check if any dates should be exempted from the calculation

   One way is to check how many weekends/public holidays exist between the start and end. We'll have to get the dates of all the Sundays and Public Holidays. If any of those dates are between the start and end, then we minus it from the # of days previously calculated in (3). Note, this step might take quite some time since we'll have to apply the same calculation to both pickup -> 1st attempt & 1st attempt -> 2nd attempt so do test it on a sample set first

5) Compare the # of days with the respective SLAs based on the SLA matrix

   If the # of days is more than the SLA, it is deemed late. Both the 1st delivery attempt and 2nd delivery attempt have to be on time, if either is late, then the whole order is considered late.