Instacart Model

Joey Sham

November 14, 2017

1 Introduction

The following document is a detailed description on the methods used to build the model for the Instacart Kaggle Competition, as well as motivation for features.

2 How It Works

2.1 Given Files

There were 7 files given:

- `aisles`
- `departments`
- `order_products__prior`
- `order_products__train`
- `orders`
- `products`
- `sample_submission`

The `orders` file lists every user, and every order each user has made. There are detailed information such as the day-of-the-week the order was placed, and the sequence of the orders. This allows insight on which products were ordered recently, and which were ordered some time ago.

The files `order_products__prior` and `order_products__train` are the exact same format, as they are from the same set of data that has been divided up to *prior* and *train*. The `order_products__*` files has detailed information on each order, including what products were ordered in each order, the order in which they were placed into the cart, and a binary value on whether each product has been ordered previously. These files can be joined with the `orders` file, as `orders` has column `eval_set` with values *prior* or *train* to indicate how the join should happen.

In this analysis, we will not be using `aisles` and `departments`.

To get all of the files, a new method called **readFiles** from the file `pandabase.py` was created to read all the files from the **/data** folder. This **/data** folder was not included in the repository due to memory limits.

2.2 Initial Analysis

The files are arranged in a very tabular manner, with strong similarity to relational database tables. That means if further analysis is requirement, it may be wise to load the information into a SQL database. Due to time constrant, the files were just loaded in memory into **DataFrames** via **pandas** in **python3**.

Even after loading the data, it is obviously that the given fields are not enough to create a model. Thus, feature engineering is required which involves creating new columns in the dataframe based on existing data. Features are divided into two sections: user-based features, and product-based features.

2.3 Feature Engineering

Before any new columns are created, a new DataFrame called `orders_details__prior` was created by joining/merging `order_products__prior` and `orders` on `order_id`. This dataframe is a detailed view of all users and all their orders, with information on which products occured in which order.

A new feature called `no_of_times_user_bought_item` is created in `orders_details__prior` that counts the total number of times a user has ordered that specific item.

2.3.1 User-based Features

The following is a list of each feature generated based on user behaviour, with a description of the meaning of each feature to each user. A SQL interpretation of this is *GROUP BY user_id*.

total_orders
The total number of orders each user has made. Instead of counting, since `order_number` exists, this is the equivalent to the **MAX(**`order_number`**)**

total_days_between_orders
Sums up the number of days between orders, to get a number representing the total number of days between orders for each user

avg_days_between_orders
Takes the mean of number of days between orders, to understand on average how long it takes for each user to make a new order

reorder_ratio
For each user, the `reorder_ratio` is calculated with

$$\begin{aligned}&\frac{number\ of\ reorders\ in\ orders}{number\ of\ products\ ordered\ that's\ not\ in\ the\ first\ order}\\ &= \frac{count\ all\ orders\ where\ reordered\ == 1}{count\ all\ where\ order\_number > 1}\end{aligned} \tag{1}$$

total_products
Counts the number of products each user has purchased. This is the equivalent to a **COUNT()** in SQL

distinct_products
Counts the number of distinct/unique products each user has purchased.

This is the equivalent to a **COUNT(DISTINCT())** in SQL

average_no_items_per_order
For each user, the `average_no_items_per_order` is calculated by

$$\frac{total\_products}{total\_orders} \tag{2}$$

which calculates the average number of items in the cart when the user orders

2.3.2 Product-based Features

The following is a list of each feature generated based on products, with a description of the meaning of each feature to each product. A SQL interpretation of this is *GROUP BY product_id*.

no_purchased
The number of times this product has been purchased by users. Note that this count is non-unique, but rather it is the equivalent to a **COUNT()** in SQL

no_reordered
The number of times this order has been reordered. Since `reordered` is a binary of 0 or 1, this sums the column to get the count.

no_bought_first_time
The number of times a product is bought the first time by a user. An interpretation of this is the number unique users has bought this item. This counts the number of times `no_of_times_user_bought_item` == 1

no_bought_second_time
The number of times a product is bought the second time by a user. An interpretation of this is the number of unique users that has bought this item a second time. This counts the number of times `no_of_times_user_bought_item` == 2

reorder_prob

This describes the probability that the product will be reordered for a second purchase after a user has bought it the first time. This is calculated by

$$\begin{aligned} & \frac{number\ of\ times\ this\ item\ has\ been\ bought\ a\ second\ time}{number\ of\ times\ this\ item\ has\ been\ bought\ the\ first\ time} \\ = & \frac{no\_bought\_first\_time}{no\_bought\_second\_time} \end{aligned} \tag{3}$$

reorder_ratio
This is the ratio of how many reorders for each item, compared to number of orders. This is calculated by This describes the probability that the product will be reordered for a second purchase after a user has bought it the first time. This is calculated by

$$\begin{aligned} & \frac{number\ of\ reorders}{number\ of\ purchases} \\ = & \frac{no\_reordered}{no\_purchased} \end{aligned} \tag{4}$$

avg_no_times_ordered
This is the average number of times an item is ordered. This is calculated by

$$\begin{aligned} & \frac{no\_purchased}{no\_bought\_first\_time} \\ = 1 + & \frac{no\_reordered}{no\_bought\_first\_time} \end{aligned} \tag{5}$$

2.3.3 User-based and Product-based Features

The following is a list of each feature generated based on both users and products, with a description of the meaning of each feature to each user and each product. A SQL interpretation of this is $GROUP\ BY\ user\_id, product\_id$.

no_of_orders
This shows the number of times each user has ordered each product

order_number_of_first_purchase

For each user, this displays which order number in their history that they first ordered an item. For example, user with id 1 ordered product 10326 the first time during user 1's 5th order. This can be calcaulted by taking the MIN of `order_number`.

order_number_of_last_purchase
For each user, this displays which order number in their history that they last ordered an item. For example, user with id 1 ordered product 10326 the last time during user 1's 5th order. As this is the same user from the example above, it can seen that he/she has only ordered this item once. This can be calcaulted by taking the MAX of `order_number`.

avg_cart_order_number
For each user, this describes the average order the product is added to cart.

order_rate
This describes how often the user orders this product. This is calculated by

$$\frac{no\_of\_orders}{total\_orders} \tag{6}$$

no_of_orders_since_last_purchase
This describes how many order it has been since the user has last ordered this specific product. This is calculated by

$$total\_orders - order\_number\_of\_last\_purchase \tag{7}$$

order_rate_since_first_purchase
This describes how many order it has been since the user has first ordered this specific product. This is calculated by

$$\frac{no\_of\_orders}{total\_orders - order\_number\_of\_first\_purchase + 1} \tag{8}$$