Instacart Market Basket Analysis

The goal of this exercise is to suggest to Instacart customers items that they might want to purchase based on past orders. The point is not to create an award-winning algorithm, but to place yourself in the shoes of a data science consultant.

1. Data Exploration
    1. What initial insights can you get from a first exploration of the dataset?
    2. Do some variables seem to have more importance than other? What transformations might be needed?

2. Prediction
    1. Describe your overall approach: how did you formulate the problem to make recommendations?
    2. What model did you choose? Why?
    3. How did you assess performance of your model? Which metrics seemed particularly important to you?
    4. Please present your results and model performance
    5. How would you suggest delivering the recommendations to your client?

3. Insights and Next Steps
    1. What insights can you extract from your analysis?
    2. What are some of the things you would suggest as next steps for your client?

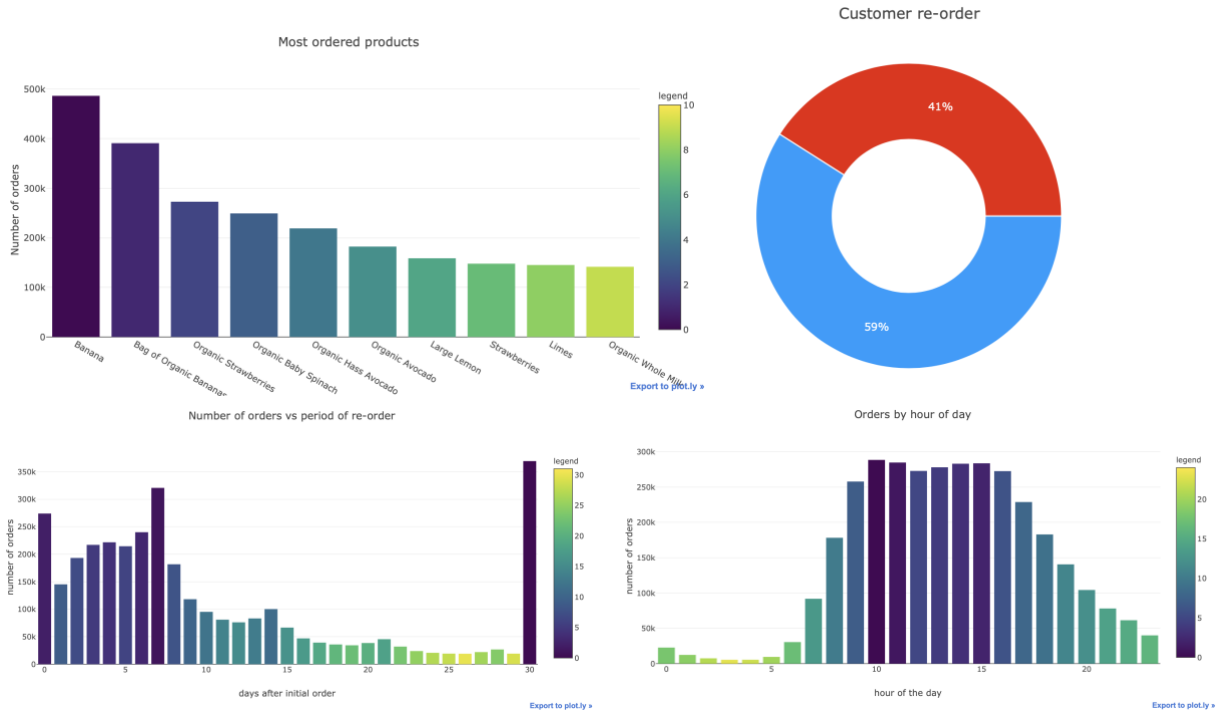
The dataset used was Instacart's open sourced user's transactional data - The Instacart Online Grocery Shopping Dataset 2017. Also, a famous Kaggle competition of 2017. The dataset is a relational set of files describing customers' orders over time. The dataset is anonymized and contains a sample of over 3 million grocery orders from more than 200,000 Instacart users.

Approaching the problem

I will implement Exploratory Data Analysis to learn more about the data and its underlying behavior and generate visualizations for the same. I will also carry out descriptive modeling for understanding customer behavior and patterns on product reorder. Lastly, I will implement a recommendation system using Apache spark to recommend to the customer the most likely products they will buy next time.

Exploratory Data Analysis

I performed basic EDA on the training data and have come up with the following initial insights –



1. About 6.4% of the columns in the prior orders contained null values.
2. The most ordered products were banana, strawberries, avocado, limes, milk etc.
3. Around 59% percentage of all products in orders is reordered at some point.
4. Chocolate, Milk, Energy drink, Banana etc. had the highest reorder probability (with a minimum threshold of 100 reorders)
5. Peak hours for online shopping tends to be around 10 am to 3 pm.
6. Most popular (number of orders) department is Produce and Personal Care has the greatest number of distinct products.
7. Customers also tend to keep recurring reorders either at an interval of 7 days or 30 days.
8. I observed a right skewed distribution for number of orders vs number of products. Most orders have 4-8 different products in the cart.

Modeling

Feature Engineering

To carry out feature engineering for predicting whether a product in an order will be reordered or not I decided to split up features into three main sections –

User features –

1. user_reorder_ratio – what ratio of orders by this user were reordered
2. user_basket_sum – sum of total number of products in basket
3. user_basket_mean – average number of products in basket
4. user_basket_std – standard deviation of number of products in basket
5. user_avg_days_between_orders – average number of days since prior order
6. user_number_of_orders – total number of orders by the user
7. user_total_items – total number of products ordered by the user

8. user_all_products – all product ids
9. user_avg_per_cart – average number of products per order

Product features –

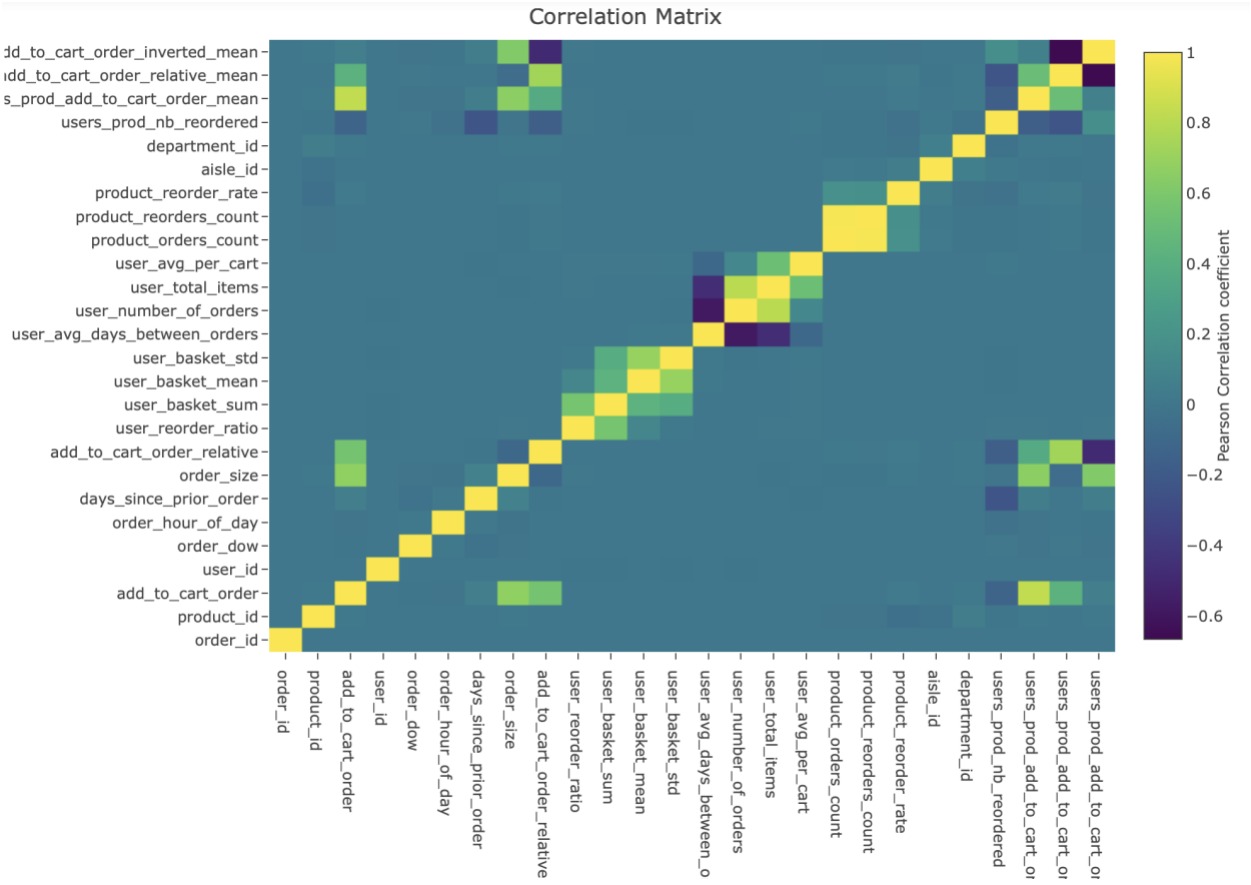
1. product_orders_count – number of orders made for the product
2. product_reorders_count – number of reorders made for the product
3. product_reorder_rate – reorder rate for the product
4. aisle_id
5. department_id

UserProduct features –

Generating user-product features by myself consumed a lot of processing time (to create user-product tuples). So, I chose to pick a feature processing method that had been used by others to solve the same problem to see the impact they could have in this case.

1. users_prod_nb_reordered
2. users_prod_add_to_cart_order_mean – add to cart position
3. users_prod_add_to_cart_order_relative_mean – mean add to cart position for that user - product
4. users_prod_add_to_cart_order_inverted_mean – inverted mean of the add to cart position
5. users_prod_last_order_number
6. users_prod_first_order_number
7. users_prod_last_order_date – last order date
8. users_prod_first_date_number – first order date

Correlations - Highly correlated features were removed at this stage.



1. There was negative correlation for User's average days between orders to both numbers of orders and total products ordered.
2. There were also visible direct correlation between reorder rate to number of orders. As more you order from Instacart you tend to keep reordering more as well.

Sampling – In order to speed up further processing, I took out 1M random samples from the ~33M reorder dataset.

Normalization – MinMaxScaler was used to scale all the numerical features.

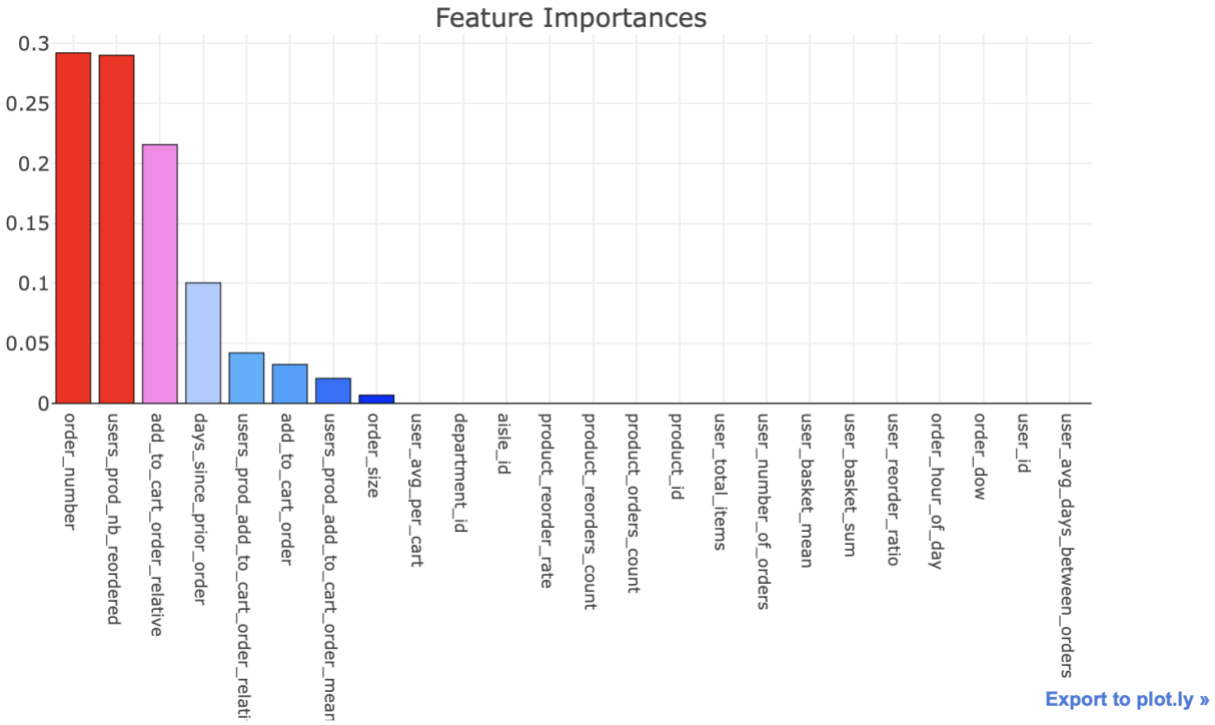
Hyperparameter Tuning – Implemented 3-fold GridSearch over 6 parameter combinations. The best performing parameters were passed on to the final XGBoost Classifier.

XGboost Classifier - XGBoost stands for eXtreme Gradient Boosting. It is a scalable and accurate implementation of gradient boosting machines. My reason to choose XGboost was its faster training speed and higher efficiency, lower memory usage and better accuracy. I also preferred LightGBM but it forced my jupyter kernel to shut down every time I tried to fit the mode.

Variable Importance – Inbuilt methods from XGBoost was used to determine the variable importance. As per the algorithm the most important features were –

1. order_numer
2. user same product reorder number

3. add to cart order relative
4. days since prior order
5. user product add to cart order relative
6. add to cart order
7. size of the order



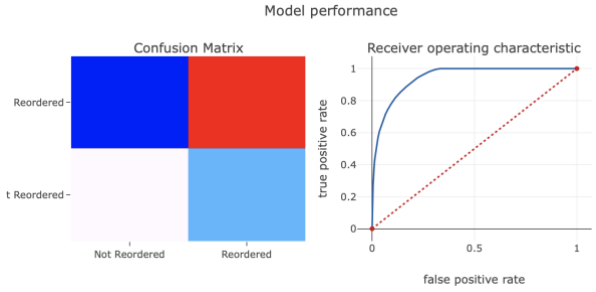
Result Metrics

```
Classification report :
              precision    recall  f1-score   support

           0       0.94      0.74      0.83     40856
           1       0.84      0.97      0.90     59144

   micro avg       0.88      0.88      0.88    100000
   macro avg       0.89      0.85      0.87    100000
weighted avg       0.88      0.88      0.87    100000

Accuracy    Score :  0.87505
Area under curve :  0.8545396819307188
```



The XGBoost model has an overall accuracy of 0.87 and an area under the curve of 0.85. The slightly higher f1 score for 1 shows that the model predicts for reorder '1' better in comparison to predicting whether a product in an order was not re-ordered '0'.

Recommendation Engine

In order to implement this, I read few materials based on building recommendation models. Couple of the main priorities that I assigned for the recommendation model were -

1. Using technologies such as Spark makes stream processing faster.
2. Generate recommendation with speed and in a distributed framework.
3. Use minimal code for the recommendation model.

After surfing through collaborative filtering section from MLlib, eventually a post from databricks caught my attention. They implemented the FP-Growth algorithm for building recommendations and explains how algorithms and infrastructure is necessary to generate your association rules on a distributed platform with the ever-growing ecommerce data. The final implementation was on scala, but I took inspiration from that article and moved forward to build the recommendation engine for this challenge.

Association rule learning is a rule-based machine learning method for discovering interesting relations between variables in large databases. It is intended to identify strong rules discovered in databases using some measures of interestingness. To prepare our data for spark processing, we will organize the data by shopping basket. Which means that, each row of our DataFrame represents an order_id with each item's column containing an array of items ordered within that order.

FP-growth is a program for frequent item set mining, a data mining method that was originally developed for market basket analysis. Frequent item set mining aims at finding regularities in the shopping behavior of the customers of supermarkets, mail-order companies and online shops. FP-Growth is an improvement of apriori designed to eliminate some of the heavy bottlenecks in apriori.

In particular, it tries to identify sets of products that are frequently bought together. Once identified, such sets of associated products may be used to optimize the organization of the offered products on the shelves of a supermarket or the pages of a mail-order catalog or web shop, may give hints which products may conveniently be bundled, or may allow to suggest other products to customers.

Analysis

Confidence and Lift measures are used to determine to what level the association rules are valid. A better value for both these scores would essentially put more trust in the subsequent recommendation model.

Results
Order Number 2115

Last order - ['Organic Mixed Vegetables', 'Organic Broccoli Florets', 'Cheese Pizza Snacks', 'Organic Spring Mix Salad']
Recommendation - ['Bag of Organic Bananas', 'Banana', 'Organic Strawberries', 'Large Lemon', 'Organic Raspberries', 'Organic Baby Spinach', 'Organic Hass Avocado']

Clearly this person like Organic foods and the model ends up recommending other organic food items.

Order Number 904
Last Order - ['Cup Noodles Chicken Flavor', 'Zero Calorie Cola']
Recommendation - ['Soda']

Future Suggestions

1. Feature Engineering – Exploring ability to process more features using word2vec from product, department texts. Having access to more hardware resources will allow generating more user-product features.
2. Using more powerful Neural Networks.
3. Using feature interactions for descriptive analysis and reorder prediction.
4. Build a front end tool, implement A/B testing to determine the success of the Recommendation model.