Instacart Market Basket Analysis Report

Yilin Du

9/3/2017

I. Definition

Project Overview

On-line shopping is not a new topic nowadays. Everyone has experience purchasing something through Amazon or Ebay. It changes our life dramatically and we can get almost everything through a simple click. To those online business owners, high repeat customer rate is the key to the success. Typical online store gets 43% of its revenue coming from repeat customers [1]. The field of repeat customer prediction is becoming increasingly popular these days. By analyzing the purchase history data of customers, the business owners will be able to predict the products he/she will purchase again. This information is extremely useful for a recommendation system, helping increase customer retention. Popular machine learning approaches include Collaborative Filtering and Content-based Filtering. The former one is based on the similarities between users while the latter focuses on description of the item and a profile of the users' preferences [2]. The project discussed in the report is a competition in Kaggle where we try to predict the user reorder behavior from a grocery deliver App.

Problem Statement

The project for the capstone is a Kaggle Competition: Instacart Market Basket Analysis. Instacart is a grocery ordering and delivery app. Customers can select products through the Instacart app and a nearby personal shoppers will review the order and do the in-store shopping and delivery. Instacart open sourced 3 million Instacart orders from more than 200,0000 users. In this competition. I use the anonymized data on customer orders over time to predict which previous purchased products will be in a user's next order.

The task can be formulated as a binary prediction: given a user, a product (this product has been purchased by the user in the previous orders) and the user's prior purchase

history, predict whether the given product will be reordered in the user's next order. The output format will be: user's last order id and a list of products that will be reordered.

Metrics

In this competition, the submission file is the order_id and a list of predicted re-ordered products in this order. The F1 score for each order is calculated between the set of predicted re-ordered products and the set of true re-ordered products. The mean F1 score for all orders will be used as evaluation metric. F1 score is defined as below:

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

In this problem, the precision is defined as the fraction of true reordered products among all predicted reordered products. The recall is defined as the fraction of predicted re-ordered products that are indeed be reordered among all true reordered products.

Using F1 score as an evaluation metric here help get a more realistic measure of the classifier's performance. It takes both precision and recall into consideration and we can avoid being fooled by unbalanced data with high precision and poor recall or the other way arount.

II. Analysis

Data Exploration

The datasets for the competition is provided by Instacart in 7 csv files, including sample of over 3 million grocery orders from more than 200,0000 Instacart users. For each user, the dataset contains between 4 and 100 of their orders, which the sequence of products purchased in each order. The median for the total orders for the users is 11 orders.

There are 206,209 customers in total. The last purchase of 131,209 users are given as training set and we need to predict for the test 7,5000 users. We need to predict the reordered products in a user's next order. On average, about 59.8% of the products in an order are reordered products, which are quite high.

All the datasets provided are in good shape and I don't see any outliers. "day_since_prior_order" is integer in [0:30], all the orders that are placed for more than 30 days are rounded as 30, this is something I need to treat specifically in the model.

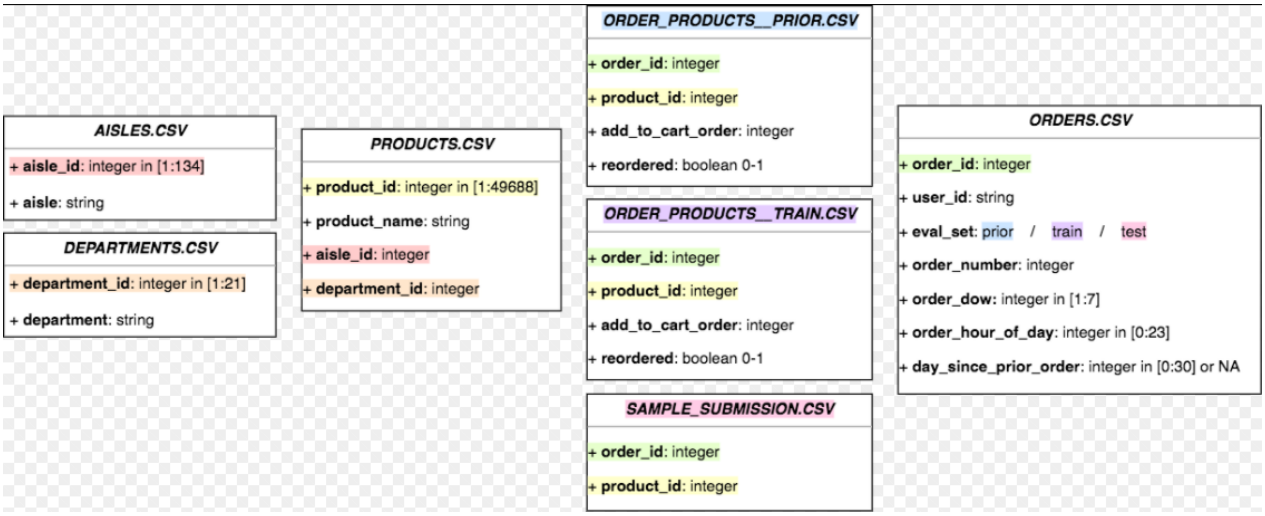


Figure 1 Raw datasets and relation between different files [3]

Below is the full data schema [4]

orders (3.4m rows, 206k users):

- order_id: order identifier
- user_id: customer identifier
- eval_set: which evaluation set this order belongs in (see SET described below)
- order_number: the order sequence number for this user (1 = first, n = nth)
- order_dow: the day of the week the order was placed on
- order_hour_of_day: the hour of the day the order was placed on
- days_since_prior: days since the last order, capped at 30 (with NAs for order_number = 1)

products (50k rows):

- product_id: product identifier
- product_name: name of the product
- aisle_id: foreign key
- department_id: foreign key

aisles (134 rows):

- aisle_id: aisle identifier
- aisle: the name of the aisle

deptartments (21 rows):

- department_id: department identifier
- department: the name of the department

order_products__SET (30m+ rows):

- order_id: foreign key
- product_id: foreign key
- add_to_cart_order: order in which each product was added to cart
- reordered: 1 if this product has been ordered by this user in the past, 0 otherwise

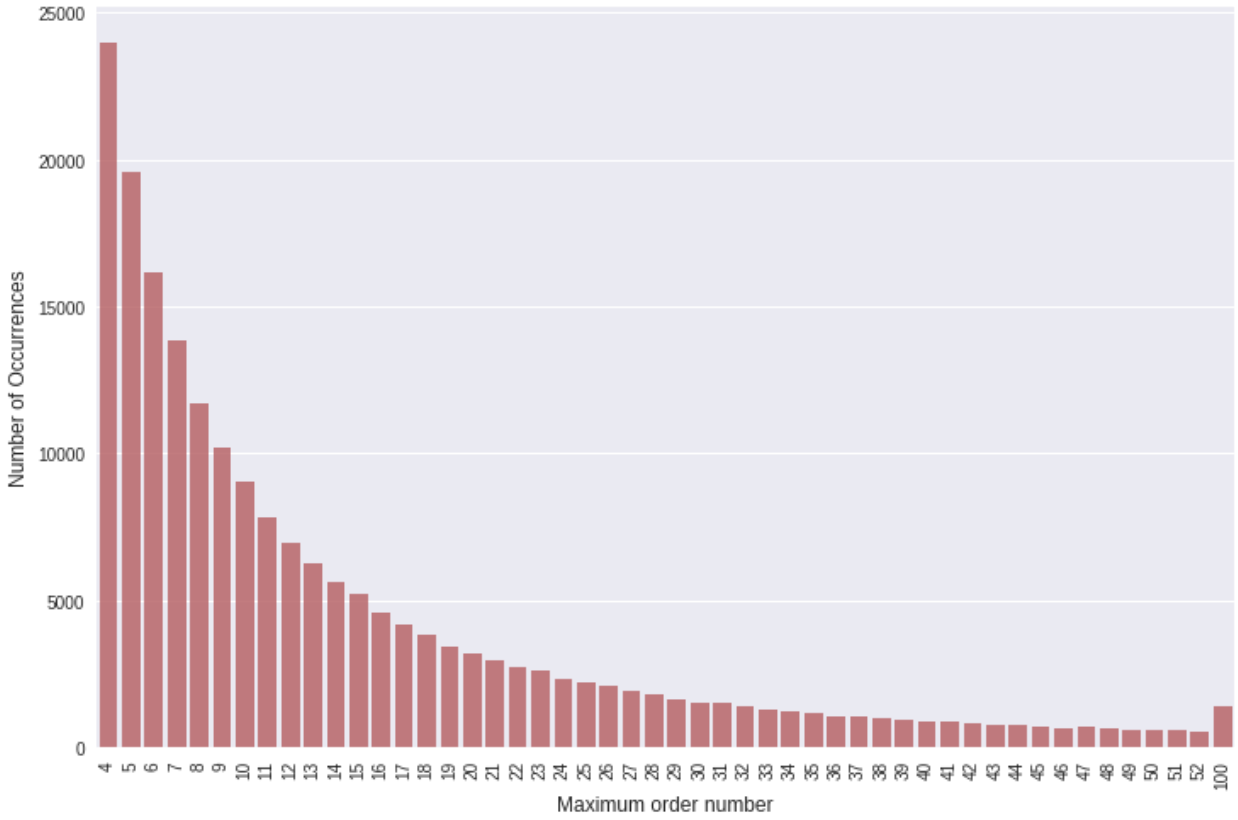
where SET is one of the four following evaluation sets (eval_set in orders):

- "prior": orders prior to that users most recent order (~3.2m orders)
- "train": training data supplied to participants (~131k orders)
- "test": test data reserved for machine learning competitions (~75k orders)

Exploratory Visualization

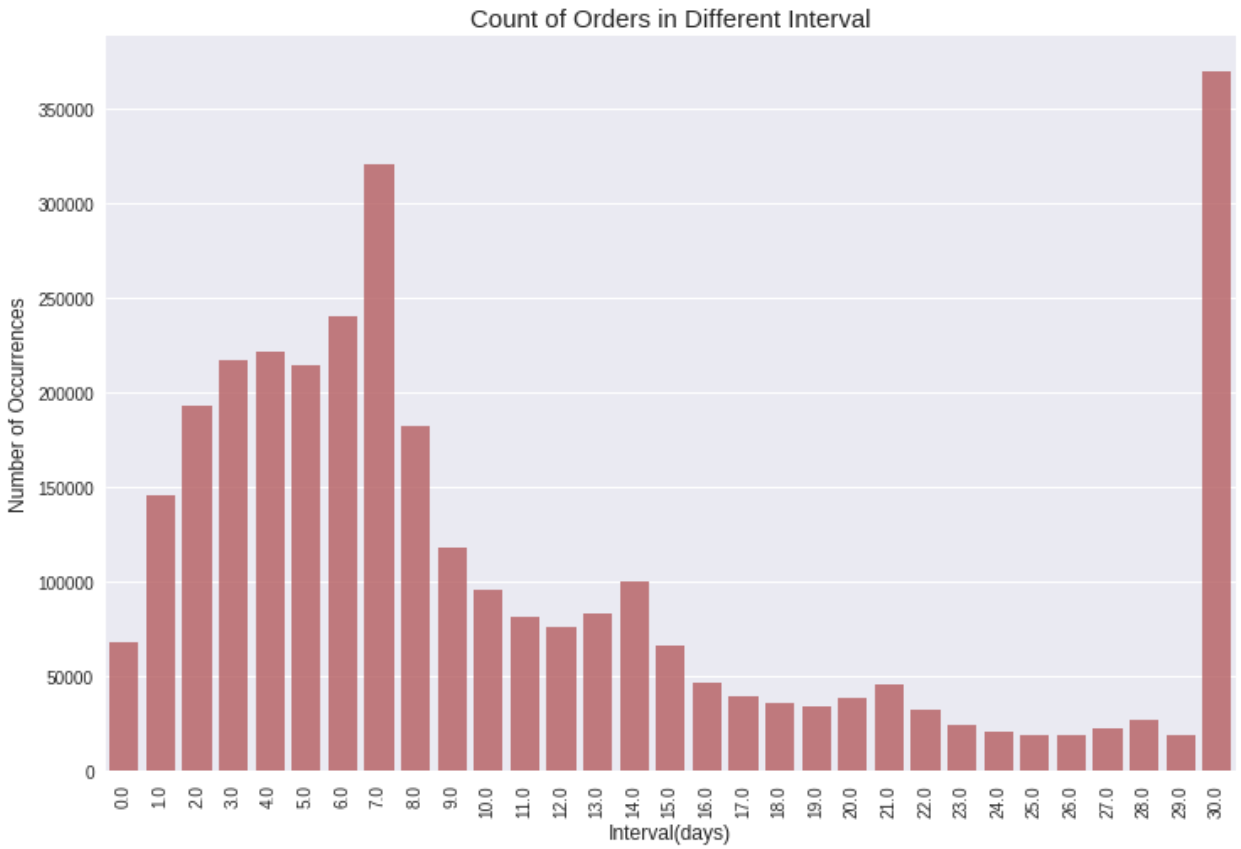
- *User Order Numbers*

  I'm expected to predict the reordered products in a user's next order based of the order histories. Let's first look at how many of the user's previous order history is provided in the dataset and its distribution. The previous 4-100 orders are provided for each user and the distribution is visualized as below (order number between 53-99 was trimmed). The median order number is 11 and the first quartile is 5 orders. We also observe a small peak for 100 orders, which indicate those loyal users for the App.



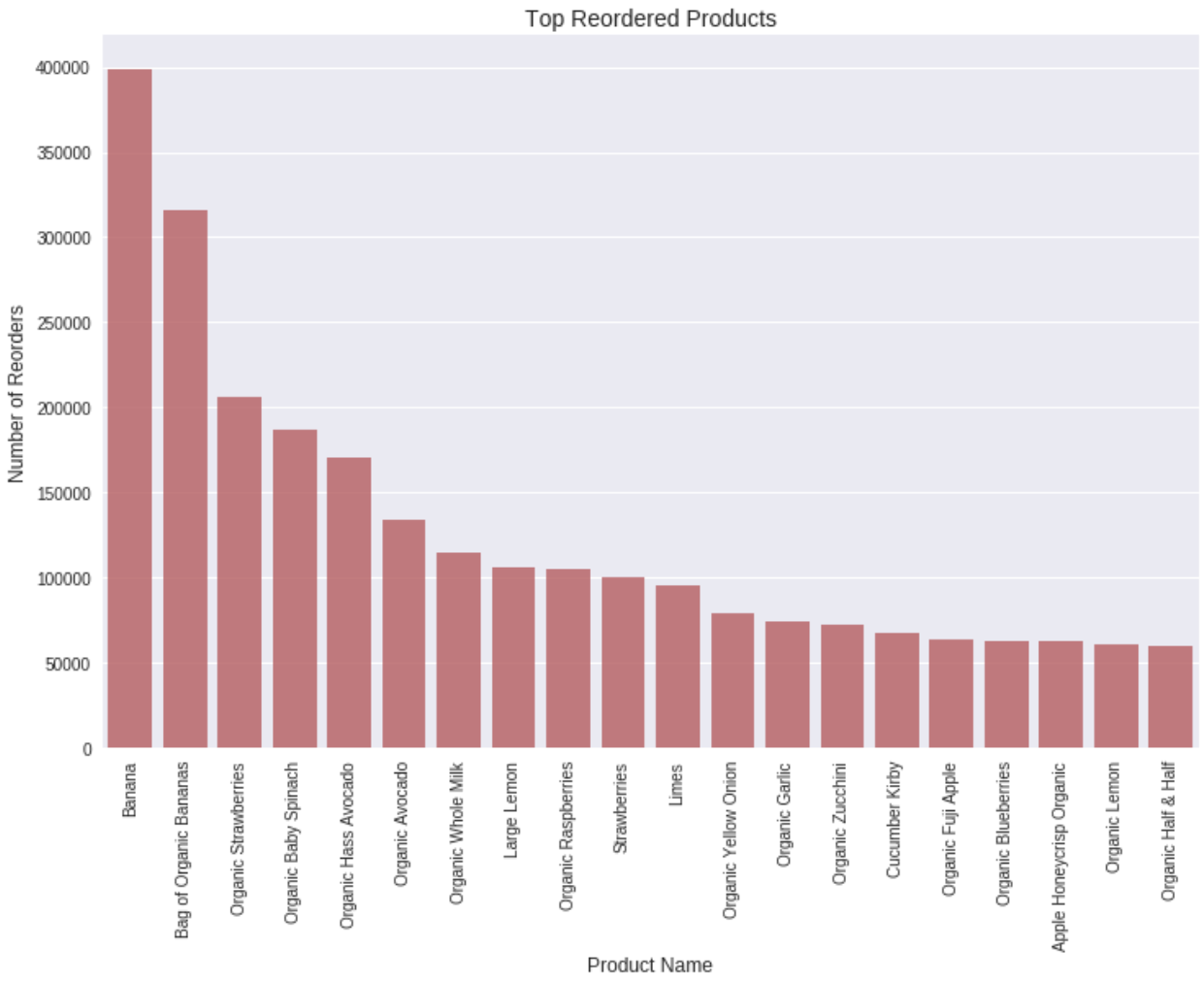
- *Intervals Between Orders*

  The interval between orders is given, and below is a distribution of it. The intervals are between 0-30days and we observe a huge spike on day 30. It should be an accumulative value for any orders that are equal or longer than 30 days. We also see small peak in days 7,14,21 and 28, which indicates the weekly order pattern by some of the users. The temporal information will be extracted as features in the model.



- *Popular Reorder Products*

  We'll predict the products that a user will order again. Let's get some idea on which products are very likely to be reordered. The figure below shows the top 20 products that have the highest reorder count by the users. Most of them are fruits and vegetables. And it seems that users from Instacart like organic products. 13 of the top 20 reordered products have key words "Organic". Some NLP process should be applied to the product names to capture the products related features.



Algorithms and Techniques

To solve the problem, the following techniques and algorithms will be applied:

Exploratory Data Analysis: Further explore the dataset to see if we have any missing data or outliers in the orders and user dataset.

Sampling: It's a huge dataset and the memory may be an issue when running the machine learning model. I sample 20% of the training set for machine learning model.

Feature Engineering: feature engineering is the key in the competition. User-related, order-related, product-related features are extracted from the dataset. Other more complicated features like user-product interaction features and NLP are applied to generate new features as well.

Machine Learning Model: I build a user-product machine learning model and apply logic regression to output the re-order probability for each product. There are many features involved in the model, popular tree-based models like xgboost are used.

A decision tree learned by splitting the source set into subsets based on an attribute value. This process is repeated on each derived subset in a recursive manner until some stopping criteria have reached. For each split, we select the attribute and cut point which leads to the greatest possible reduction in residuals sum of square. Xgboost is a tree based ensemble model. It's a stage-wise algorithm. In each stage, it introduces a weak learner to compensate the shortcomings of existing weak learners.

F1-optimization: The output of the machine learning model is a list of products the user purchased and the re-order probability for each product. I apply an algorithm to maximize the f1 score for the product list selected given their probabilities to be re-ordered.

Model Stacking: It's a kaggle open competition and some kagglers will share their model and predictions in the forum. In the final stage, I use some stacking technologies to combine my model and other well-performed shared model results for final submission.

In stacking process, we have multiple models trying to predict the target. Each model has the its own output. We will need to find a combining mechanism to combine the results. The simplest way for regression is through averaging or getting median out of all outputs. In this way, the final results filter out some outliers in each model and practically demonstrate better results.

Benchmark

For this problem, a simple and naive benchmark is used for the re-order products list for users. Each user was given between 4 and 100 previous orders. I use the second latest order each user and list all the re-ordered products in that order. Then I make a simple prediction that the user will purchase those products in the latest order. In this way, I don't use any extra information of products/users/orders but only use the re-ordered products information in the second last order as a prediction for the last order. The benchmark achieves a baseline mean F1 score of 0.304.

III. Methodology

Data Preprocessing

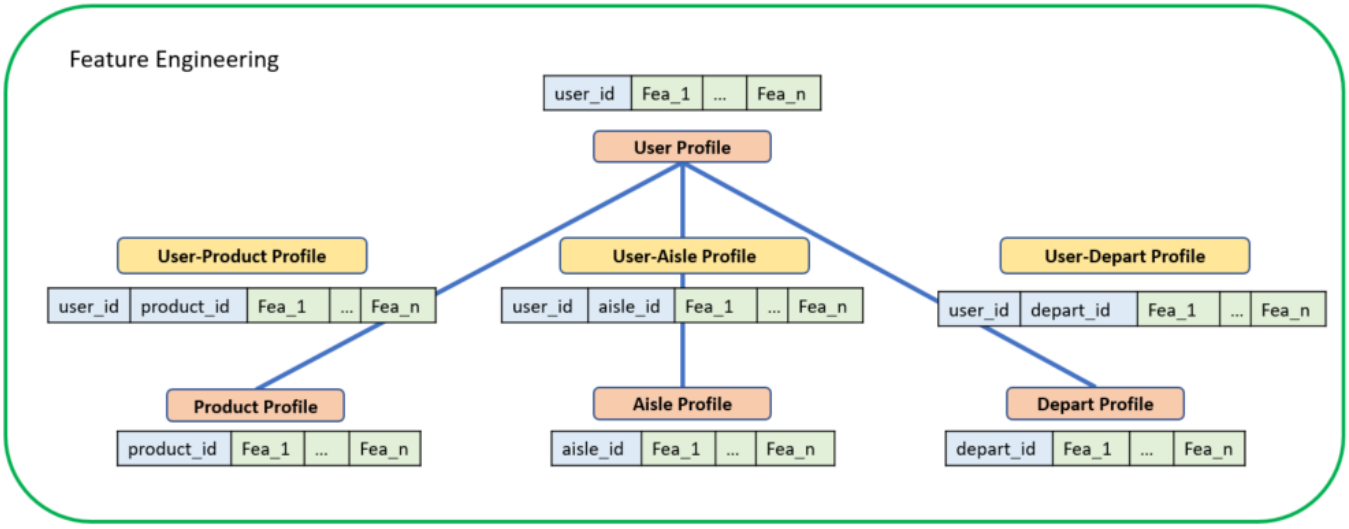
The dataset is in high quality and there is no missing data for the orders.

Due to the RAM limitation (I have 8Gb RAM), for most of the exploration parts, I sample 20% of the training dataset based on the user_id (sample from 206k users and use their previous order histories) for training.

Based on the data exploration section, the "30days" interval between orders should be treated carefully because all the orders that are made more than 30 days are marked as "30 days". I create a feature "is_30_days" to capture this information. Because this order may be made for a larger interval and might have less similarity with the previous order patterns.

Implementation

- *Feature Engineering*



Comprehensive feature engineering work is done to generate features for each user-product reorder prediction. The main profiles include user-related profile, user-related profile, aisle-related profile, department-related profile and the interaction between different profiles. Feature types include count/ratio, temporal aggregation features and PCAs. A brief summary of some sampled features is as below:

User Profile: total order number, total products number, distinct product number, average interval between order, number/ratio of weekly/weekend orders, number/rate

of reorder products, preferable order hour of day/day of week, first/last two orders size and comparison, average products number in one order, etc.

Aisle/Department Profile: total order/reorders, number of unique users/products, average interval between orders, average add-to-cart orders, etc.

Product Profile: total orders/reorders, reorder rate, average interval between orders, average add-to-cart orders, etc.

Interaction Profiles: aggregation metrics generated from both profiles.

I also add some complicated features into my model inspired by the other kaggler's shared ideas in the forum.

Word2Vec for product recommendation [5]: Word2Vec is an unsupervised algorithm that learns vector representation of words from a huge corpus of text. In this competition, we apply the Word2Vec model on the data of product orders where the orders act as a long sentence and product ids acts as words. The model outputs a vector representation of each product id (the vector is set to be size 100 in the model). Then I constructed 3 compressed feature using PCA on the vector representation matrix for each product id.

User TFIDF feature [6]: We treat each product name as a term and for each user, calculate the term frequency in each user's order history. TFIDF will pick the top 1000 vocabulary (e.g. "organic", "bananas", "milk"…) and transform them into a vector. Since the output from TFIDF is quite sparse, so we use truncated SVD to generate 2 components for each user.

I generated more than 100 features for my model. For feature selection, I use simple LEAVE-ONE-OUT strategy to select 92 features for the final model. For the 10+ features I discard, the F1 score in local cross validation doesn't improve or even worsen when adding them.

- *XGBoost Modeling*

For the parameter tuning, this blog post gives a good recommendation on the parameters setting [7].

Due to the number of features and the size of the dataset, it takes more than 1 hour for me to train the model. I use grid search on a few key parameters in the XGBoost setting: eta (0.03, 0.05, 0.1, 0.3, 0.5), max_depth(4, 6, 8, 10), lambda (1, 5, 10, 20). The parameter

tuning part for XGBoost is challenging and requires further work if someone try to re-implementing what I have done. I selected the search space based on some online post suggestions. But a more robust and systematic method should be applied for parameters tuning.

The parameters I use for the final model is as below:

"objective" : "reg:logistic"
"eval_metric" : "logloss"
"eta" : 0.1
"max_depth" : 6
"min_child_weight" :10
"gamma" :0.70
"subsample" :0.76
"colsample_bytree" :0.95
"alpha" :2e-05
"lambda" :10

- *Threshold Selection*

order_id	product_id	pred
1	21903	0.0234
1	17668	0.4532
1	39190	0.2739
...		

order_id	products
1	17668 39190 14987 21340 23480 4595
2	22008
3	None

The logistic regression model outputs the probability that each previously purchased product be reordered in the next order. However, it's a binary classification problem evaluated by F1 score. So, we need to come with some threshold to select the reorder products based on the predicted probabilities. For the first attempt, I use same threshold (0.2) to each order determined by grid search. If none of the product in the order has a predicted probability higher than threshold, we predict "None" to that order.

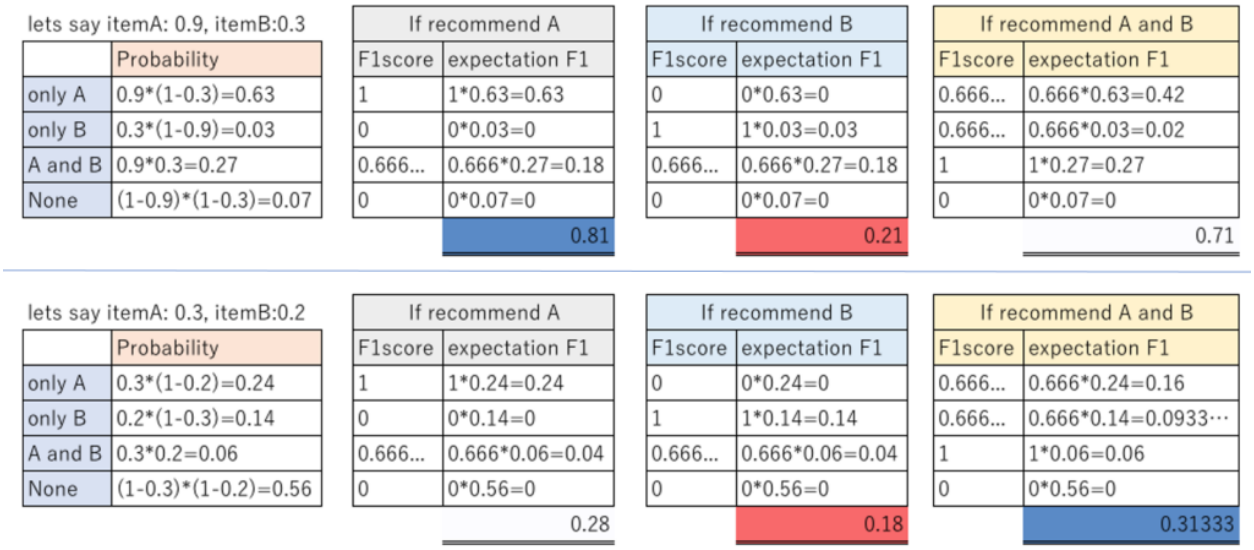
Refinement

- *F1 Optimization*

  The first attempt is to use same threshold for each order to select the products we want to predict to be reordered. But this naïve approach won't guarantee the optimization of F1 expectations given the probability.

  Here is a simple demonstration [8]

  For a simple case that we only have products A and B. First case: A has a probability of 0.9 to be reordered and B has a probability of 0.3 to be reordered. Second case: A with probability of 0.3 and B with probability of 0.2. For both cases, if I use the same threshold (0.2) for products selection, I'll select A and B. But let's calculated the expectation for F1 score for only selecting A, only selecting B and selecting both. We can see for case 1, only selecting A gives the highest F1 expectation of 0.81. For case 2, selecting both will have better results. This demo indicates that for different orders, we should choose different threshold to boost the F1 score.



lets say itemA: 0.9, itemB:0.3

	Probability
only A	0.9*(1-0.3)=0.63
only B	0.3*(1-0.9)=0.03
A and B	0.9*0.3=0.27
None	(1-0.9)*(1-0.3)=0.07

If recommend A	
F1score	expectation F1
1	1*0.63=0.63
0	0*0.03=0
0.666...	0.666*0.27=0.18
0	0*0.07=0
	0.81

If recommend B	
F1score	expectation F1
0	0*0.63=0
1	1*0.03=0.03
0.666...	0.666*0.27=0.18
0	0*0.07=0
	0.21

If recommend A and B	
F1score	expectation F1
0.666...	0.666*0.63=0.42
0.666...	0.666*0.03=0.02
1	1*0.27=0.27
0	0*0.07=0
	0.71

lets say itemA: 0.3, itemB:0.2

	Probability
only A	0.3*(1-0.2)=0.24
only B	0.2*(1-0.3)=0.14
A and B	0.3*0.2=0.06
None	(1-0.3)*(1-0.2)=0.56

If recommend A	
F1score	expectation F1
1	1*0.24=0.24
0	0*0.14=0
0.666...	0.666*0.06=0.04
0	0*0.56=0
	0.28

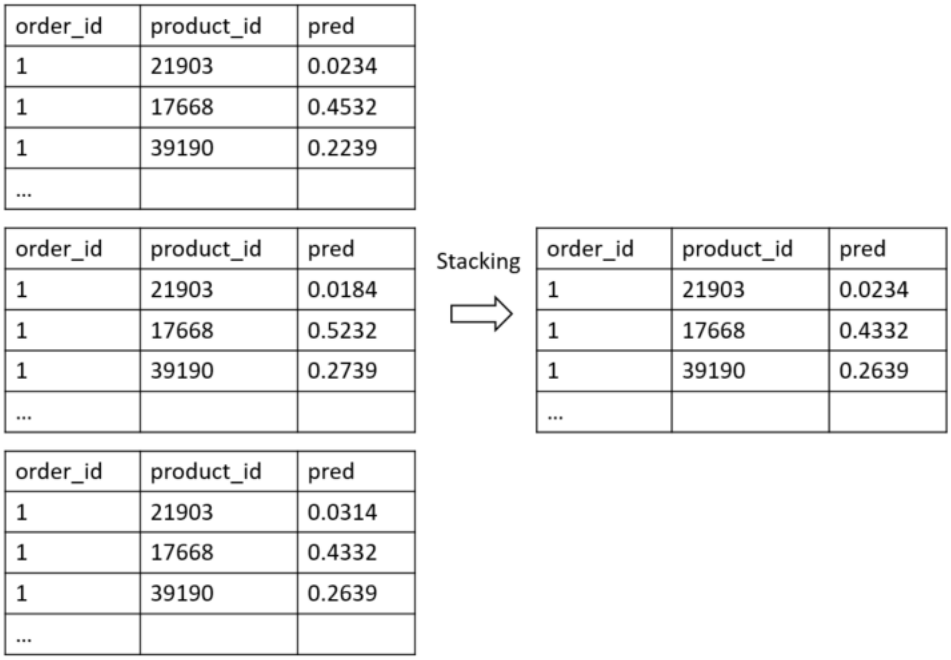
If recommend B	
F1score	expectation F1
0	0*0.24=0
1	1*0.14=0.14
0.666...	0.666*0.06=0.04
0	0*0.56=0
	0.18

If recommend A and B	
F1score	expectation F1
0.666...	0.666*0.24=0.16
0.666...	0.666*0.14=0.0933…
1	1*0.06=0.06
0	0*0.56=0
	0.31333

  The F1-score expectation maximization algorithm for this problem was presented in the paper (add reference) and the script [9] in kaggle implement the methods. I apply the script and get an int K returned for each order, which indicates I should select the top K products in the given order with highest probability.

- *Model Ensembling*

Model ensembling is a very popular and powerful technique to increase accuracy on machine learning models. At the final stage of this competition, a participant open sourced his code. I implement a simple stacking model with 2 of my prediction files (generated by using different 20% sample of training data) and the prediction generated by the open sourced code. The below diagram is a demonstration how I generate a stacking prediction file based on the median values of 3 input files. I also try using the average values of the 3, however, it gives worst results than using the median.



IV. Results

Model Evaluation and Validation

In the competition, I use 20% of random sampled training date for the machine learning model. In different submission to the competition, I try train the model with different 20% samples and the results don't vary much (~0.0005 difference in F1 score). So the input data won't greatly change the accuracy. During the competition, I summit a csv file of predicted product list in 75,000 orders. The public leaderboard is calculated with approximately 33% of the data (that's the score I see during the competition). When the competition ends, the final standings is calculated with the rest 67% of the data. In my

submissions, the private scores are always about ~0.001 lower than the public scores. So the models are robust enough and the scores I see during a competition is very close to my final standing.

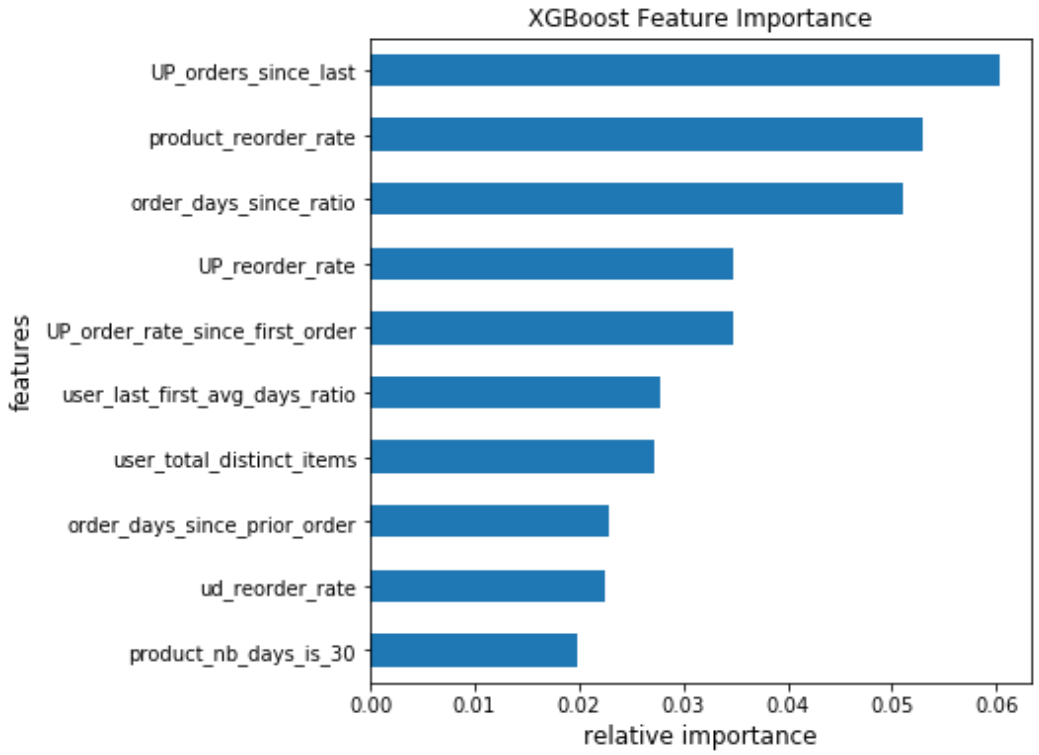
Justification

I have in total 36 submissions in the competition. Below is the table summarizes the major boosts in my F1 score. I rank 230 among 2630 participants (top 9%). And the submission of the winner has a F1 score of 0.409.

Submissions	F1 Score	Major Changes
1	0.304	Baseline
2	0.376	XGBoost with ~30 features
3	0.385	XGBoost with ~100 features
4	0.395	F1-Optimization
5	0.402	XGBoost Tuning, feature selection
6	0.403	Stacking
First Place	0.409	

V. Conclusion

Free-Form Visualization



The plot above is the top 10 importance features in the xgboost model. The most importance feature to the model is UP_orders_since_last, which is defined as the difference between the user's total order number and the most recent order number in which the user ordered the product. The reorder rate of a product is also an important factor.

Reflection

The main goal of this project is to predict whether a product will be reordered in a user's next order based on the order history of the user. The solution to the problem includes 2 stages: first, build a logistic regression model on userXproduct basis, the model outputs the probability that the product will be re-ordered in the user's next order. Second, come up with an optimization algorithm to select a group of products based on the probabilities, thus maximizing the F1 score. The specific steps to go through include: exploratory data analysis and data preprocessing, sampling, feature engineering, machine learning model, F1-optimization of model emsembling.

The most challenging and time consuming process in this project is feature engineering. I generated a large number of features to capture the preferences and behaviors of the users, characteristics of the products, aisle, department, temporal features and the

interactions among them. None of those features is a strong indicator for the predictions and more powerful features may be generated with the domain knowledge on customer behaviors.

The F1-Optimization algorithm boosted the mean F1 score quite a lot. I didn't dive deep in the mathematical derivation of the algorithm. But it is a powerful tool to deal with any similar multi-label classification problems in the future.

Improvement

There are many process in the project can be further improved:

Feature Engineering: the feature engineering process in the project is a bit messy and unorganized. I generated too many features and it's hard to evaluate which features are most important for the model training. It's better to group the large number of features and evaluate the relative importance between different groups as well as that in the same group. To generate any magic features, more research on customer behavior studies is required.

Model Selection and Tuning: I used XGBoost in this problem mainly because other anticipants shared simple starter code to this problem in the forum. I should explore the other machine learning algorithms and compare the results based on efficiency and accuracy. I did grid search for a few parameters to tune the model. More efficient model tuning techniques should be applied.

Reference

[1] https://www.bigcommerce.com/blog/the-3-most-important-e-commerce-benchmarks-and-how-you-can-crush-them/

[2] https://en.wikipedia.org/wiki/Recommender_system

[3] https://www.kaggle.com/c/instacart-market-basket-analysis/discussion/33128

[4] https://gist.github.com/jeremystan/c3b39d947d9b88b3ccff3147dbcf6c6b

[5] https://omarito.me/word2vec-product-recommendations/

[6] https://www.kaggle.com/tedchang0102/using-tfidf-as-user-s-affinity

[7] https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

[8] https://github.com/KazukiOnodera/Instacart

[9] https://www.kaggle.com/mmueller/f1-score-expectation-maximization-in-o-n