Instacart Reorder Prediction Report

Qiao Wang
Sep 19, 2022

Introduction

Instacart is one of the largest online grocery shopping websites in America. In the Kaggle Instacart Market Basket Analysis competition, Instacart provided 3 million anonymized customer order histories and asked participants to predict which products a user will buy again if any. This competition uses the F1 score as the final evaluation metric to balance precision and recall.

To finish this challenge, I designed three types of features -- user-level features, product-level features, and user-product interaction features-- to capture the characteristics of each entity.
I trained an XGBoost Classifier to predict the possibility that a user would reorder a previously bought product. I then feed the predicted probability into a script that maximizes the expected F1 score and outputs the most likely products for each user to be reordered. My final prediction on the test set reached an F1 score of 0.403, roughly at the 240th position in this competition.

During the modeling stage, I used MLflow to track model performance and log parameters. This has ensured reproducibility and traceback. After the model was built, I used an explainable AI tool, SHAP, to explain the outcomes of the XGBosst Classifier. This has helped me debunk the black box of the algorithm and better understand how the classifier makes predictions.

Data

Instacart provided 3 million order histories from 206, 000 anonymized users. For each user, the basic order log is provided. It includes the intra-order sequence, products bought in each order, aisles and departments of the products, the day of the week (Dow) and hour of the day when the order was placed and days interval between two consecutive orders. With this information, we need to predict which products each customer will reorder. Figure 1 uses the first user as an example to summarize the provided data and the (transformed) prediction task.

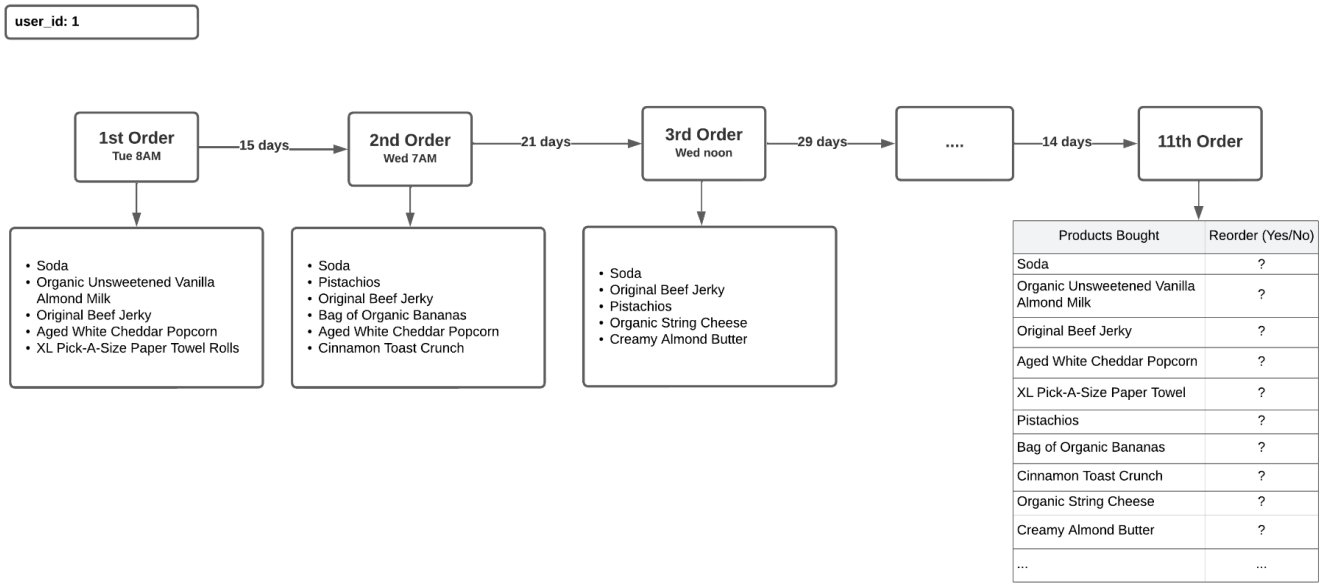


Figure 1: User Order History and Prediction Task

Exploratory Data Analysis

Figure 2: Number of Orders by Days Since Prior Order

From Figure 2, we can see that 30 days is the most common interval between two consecutive orders. However, there are local peaks each week, on the 7th, 14th, 21st, and 28th day.

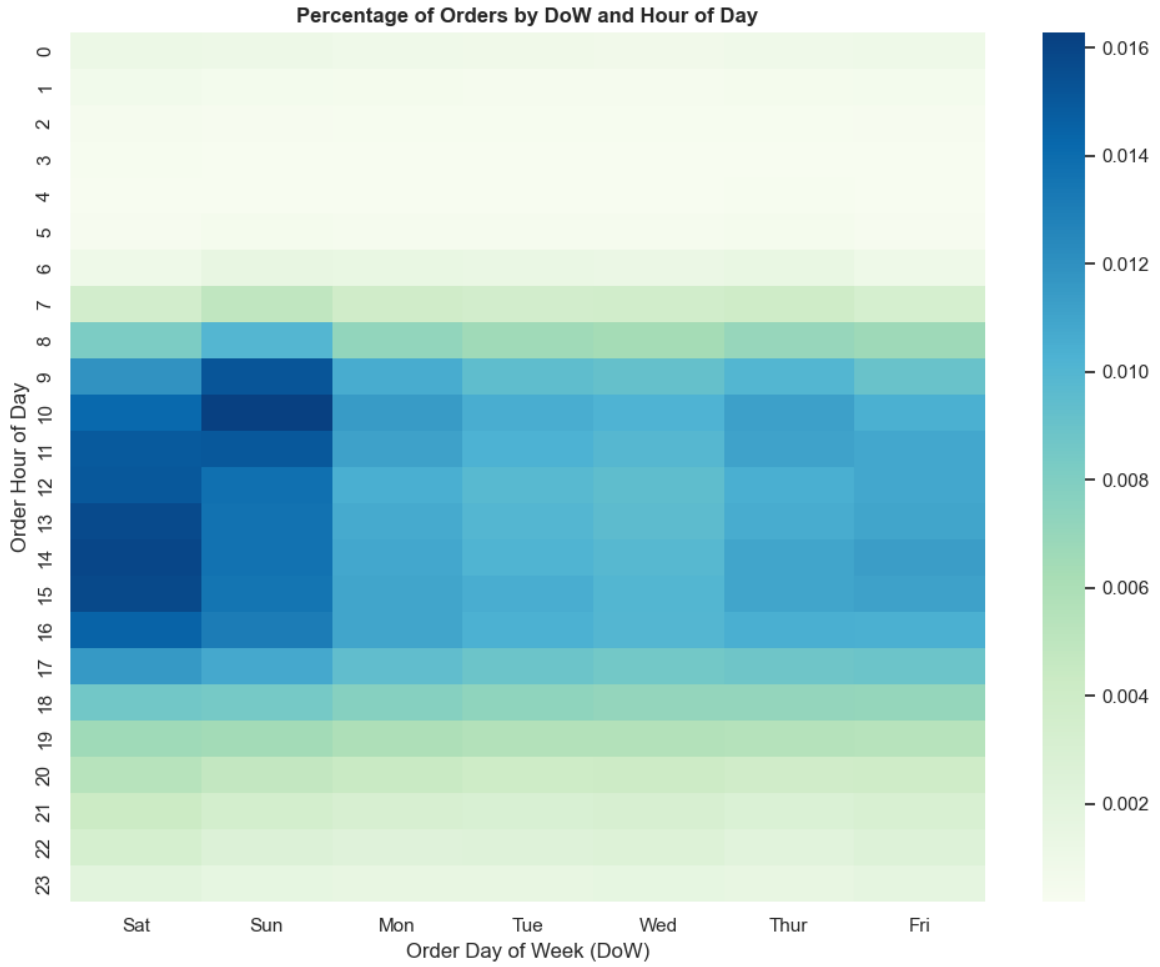


Figure 3: Order Distribution by Day of Week (DoW) and Hour of Day

Saturday afternoon and Sunday morning are the most popular time to make orders. Wednesday has the least orders in a week.

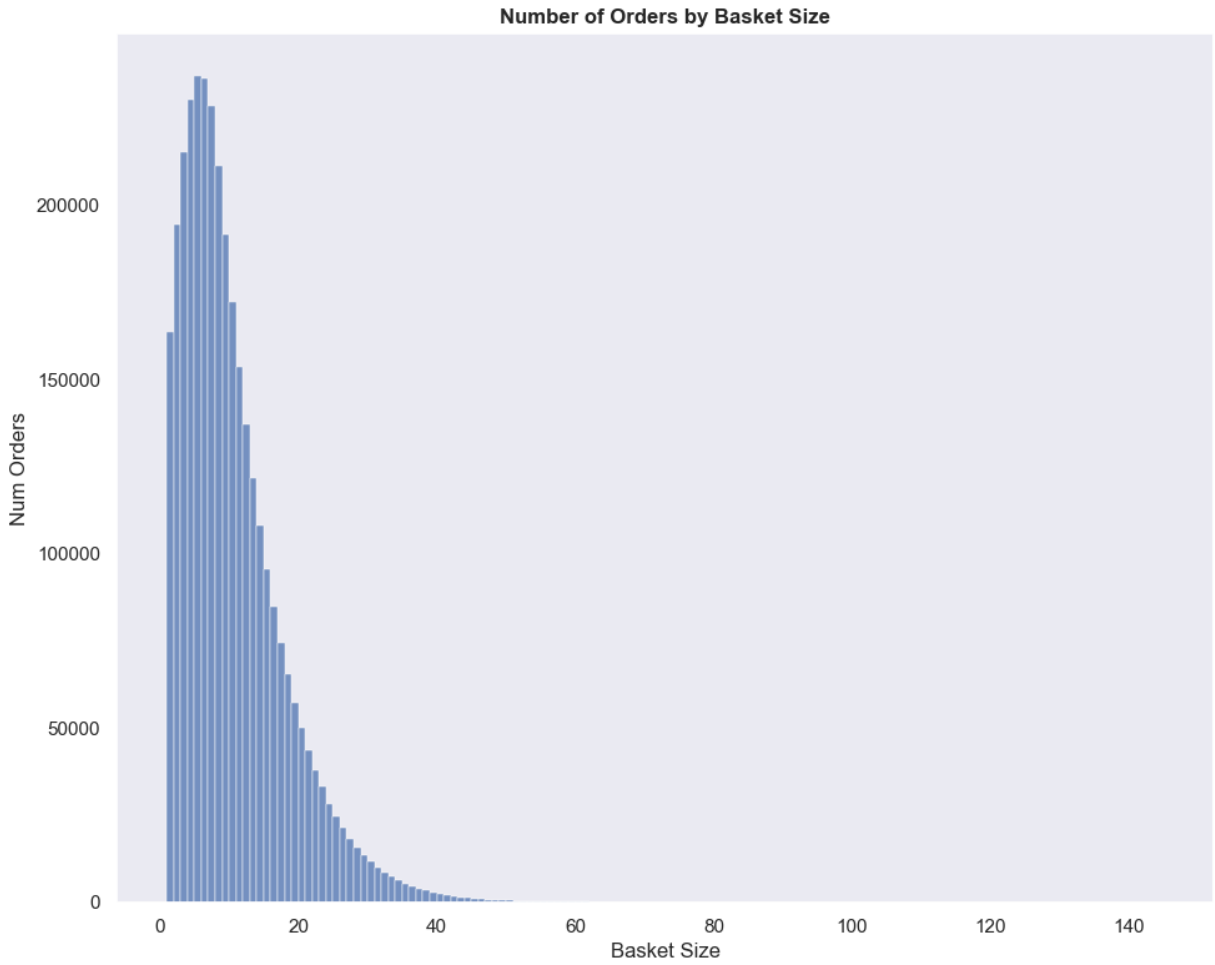


Figure 4: Order Distribution by Number of Items in Basket

Distribution of basket size is right skewed. The mode of the number of products in an order is 5. Common basket sizes are 4, 5, 6, and 7.

Top Purchased and Top Reordered Aisles

Aisle	Purchase Times	Aisle	Reorder Rate
fresh fruits	1,862,949	milk	0.78
fresh vegetables	1,486,803	water seltzer sparkling water	0.73
packaged vegetables fruits	1,229,500	fresh fruits	0.72
yogurt	880,330	eggs	0.71
milk	815,345	milk soy lactose free	0.69

...	...	...	...
eye ear care	9,136	beauty	0.21
baby accessories	8,380	first aid	0.20
baby bath body care	8,257	kitchen supplies	0.19
beauty	6,111	baking supplies decor	0.17
frozen juice	4,488	spices seasonings	0.15

Figure 5: Top Purchased Aisles and Top Reordered Aisles

Fresh fruits, fresh vegetables, and packaged vegetables and fruits are the top ordered categories while milk, water seltzer sparking water and fresh fruits are the top reordered aisles. This is maybe because there are less diverse products in milk and sparkling water aisles.

Feature Engineering

As mentioned, I designed three types of features: user-level features, product-level features, and user-product interaction feature to capture the characteristics of each entity.

User Features

User features describe what the customers look like, also known as (aka) user profiling. I created X number of user-level features to answer these questions:

- How long has the user been with us/User age and number of orders on the platform?
- How often do they place an order? and when do they usually place orders?
- What's the basket size for the user? How diverse is their basket?
- Are they more likely to reorder stuff or do they prefer trying new products?
- How long they haven't placed an order and how ready they are to place the next order based on their common purchase intervals?
- Are they interested in certain types of food, such as organic, Asian, or gluten-free food?
- Use coefficients of linear regression to measure if there is any trend/changes in users' purchasing behavior. For example, do they order more and more often, order more(less) items each time?
- Users segment features based on the KNN algorithm

Product Features

Product Features describe what the products look like. They quantify products' popularity, product stickiness, common purchase cycles, etc. I created Y number of product level features to measure:

- How popular each product is, aka, how many times they are purchased, by how many different users?
- How are they added to the shopping cart? Often added first or added very late?
- Ratio of one-time purchases or reorder rate?
- Product purchase probability: After customers tried it for the first time, what's the probability of ordering it again?
- How often products are reordered? Every week or every month? Coca-cola and toilet paper have very different purchasing intervals

User x Product Features

Interaction features measure how a user feels about a product and capture his/her purchasing behavior on a certain product. They are intended to capture users' likeness, and purchase cycles on different products. I created Z number of user x product features to answer these questions:

- How many times does a user buy a product, how many reorders and reorder rate
- What's users' purchase probability on each product? Probability is defined as after trying a product for the first time, the probability that users reorder it
- How often do users buy and reorder each product? Every week or every month?
- How long have users gone without this product?
- When do users buy this product and what's the cart position?
- Use coefficients of linear regression to measure if there is any trend/changes in users' purchase of products in the same aisle. Do they start to buy more/fewer products in the same aisle and department?
- How is users purchase on the substitution products?
- How ready are users to make the next purchase on this product, given users' purchase intervals? Essentially the difference between the last 2 metrics.
- How ready are users to make the next purchase on this product, given the general purchase cycle on this product?

It makes sense that users' recent orders are more representative, compared with more historical purchases. Therefore, for some user x product features, I also created the corresponding "recent 5" versions with the suffix "_r5" in the feature names. They are calculated based on only the recent 5 orders of users rather than the entire history orders. I introduced this recency in features because I believe users' behavior disclosed in the recent orders has more predictive power than older orders. This assumption is also verified in the feature importance ranking of the final model.

In the end, I have 165 features. I did feature selection based on the feature importance provided by the XGBoost model and only used the top 70% of the features. That's 116 features.

Model

XGBoost classifier is a boosting algorithm that builds decision trees sequentially, with each tree trying to improve slowly on the results of the last tree. It's an algorithm that has won many Kaggle competitions. I chose this model not only because of its high prediction power but also because it's parallelized and memory-efficient. In fact, one of the challenges of this prediction task is its massive dataset. Therefore it's important to use a fast and memory-efficient algorithm to speed up the experimentation cycle.

I used the XGB classifier to predict the probability of reordering for each user and previously-bought product. In other words, the entity of the training set is each pair of user id and product id. In this way, I converted the problem into a binary classification and used log loss as the optimization metric during model training.

Thresholding

Even though the XGB classifier can output the probability of re-ordering, we still need to find a way to convert the probability into a binary Yes or No answer. In convention, we can use the default 0.5 probability as the threshold. Products with a reordering probability greater than 0.5, will be reordered, and otherwise, not reordered. Of course, we can find a better threshold that maximizes the F1 score than the default value. This method is called the global threshold approach. However, my experiment shows that using different thresholds for different users, the local threshold approach, can further optimize the F1 score. To illustrate this, let's use two examples below:

P(A)	0.4
P(B)	0.3

		F1 in Difference Scenarios		
Reality	P(reality)	Predict A Reordered	Predict A & B Reordered	Predict None Reordered
A actually reordered	0.28	1.00	0.67	0.00
B actually reordered	0.18	0.00	0.67	0.00
A & B reordered	0.12	0.67	1.00	0.00
None reordered	0.42	0.00	0.00	1.00
Expected F1		0.36	**0.43**	**0.42**
Threshold Range		**(0.3, 0.4)**	**(0, 0.3)**	**(0.4, 1)**

Figure 6: First Example of Expected F1 with Different Thresholds

Let's say, for a user, the model predicts that product A will be reordered with a probability of 0.4 while product B will be reordered with a probability of 0.3. With different thresholds, the model predicts different products being reordered. There are in total three scenarios and I calculated the expected F1 score under each scenario. We can get the maximum expected F1 of 0.43 if we choose a threshold smaller than 0.3, and predict both A and B being reordered.

P(C)	0.9
P(D)	0.3

		F1 in Difference Scenarios		
Reality	P(reality)	Predict C Reordered	Predict C & D Reordered	Predict None Reordered
C actually reordered	0.63	1.00	0.67	0.00
D actually reordered	0.03	0.00	0.67	0.00
C & D reordered	0.27	0.67	1.00	0.00
None reordered	0.07	0.00	0.00	1.00
Expected F1		**0.81**	0.71	0.07
Threshold Range		**(0.3, 0.9)**	**(0, 0.3)**	**(0.9, 1)**

Figure 7: Second Example of Expected F1 with Different Threshold

For another user, the model predicts that product C and D being reordered with probability 0.8 and 0.3. Similar to the first example, we have three scenarios when we set different thresholds: only product C being reordered, both being reordered and either being reordered. As we did in the last example, I calculate the F1 score in each scenario. We will get the max F1 if we predict only C being reordered. To get this max F1, we have to set our threshold between 0.3 and 0.9, which is exclusive to the threshold range in the first example. Therefore, we have to choose different thresholds for different users to get the max F1 score.

Performance

Due to the limitation of computer memory, only 60% of data (4.5 GB) is used for training. Data are split into train and test sets based on user id and 80% of users are in the train order while 20% of users are in the validation set. The performance on the validation set is summarized in figure 8.

A precision of 0.39 means that for all the products we predict to be reordered, 39% of them are actually reordered by the customers. A recall of 0.5 means that for all the products that customers actually reordered, our model identified half of them. Combining Precision and

Recall, we have an F1 score of 0.44. There is a significant gap between F1 and accuracy because reordered products are only a minority (9%) of all products that customers previously purchased.

F1	0.44
Precision	0.39
Recall	0.50
Accuracy	0.87

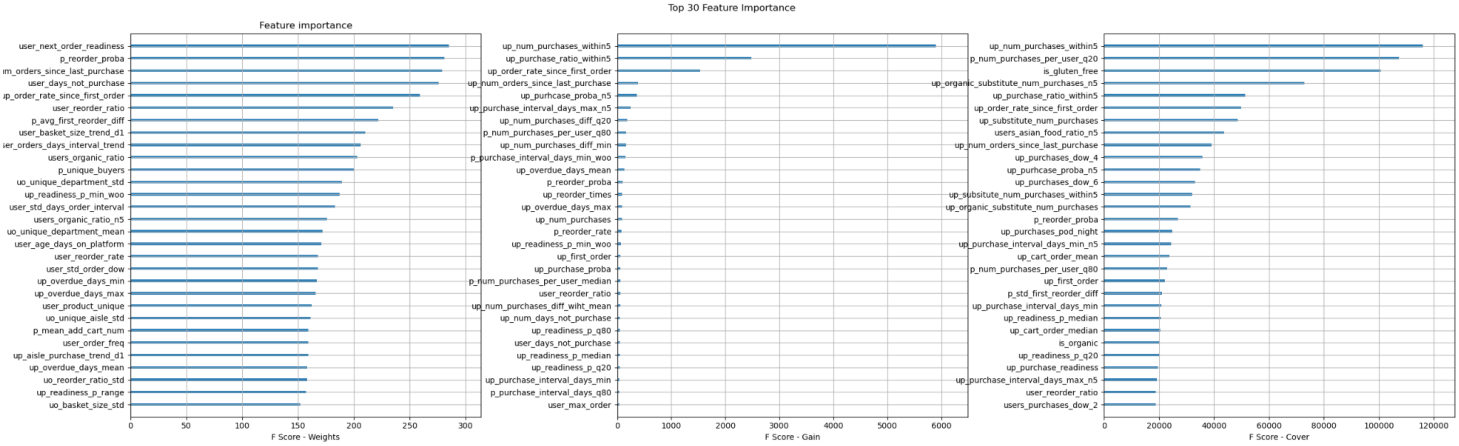
Figure 8: Test Performance

Besides validation performance, the prediction on the given test set has achieved an F1 score of 0.402 on the Kaggle Leaderboard.

Top Features and Insights

XGBoost provides three ways of measuring feature importance, weights, gain and cover. I've analyzed the top 30 features under each measurement.
For better readability, note these name conventions in feature names:

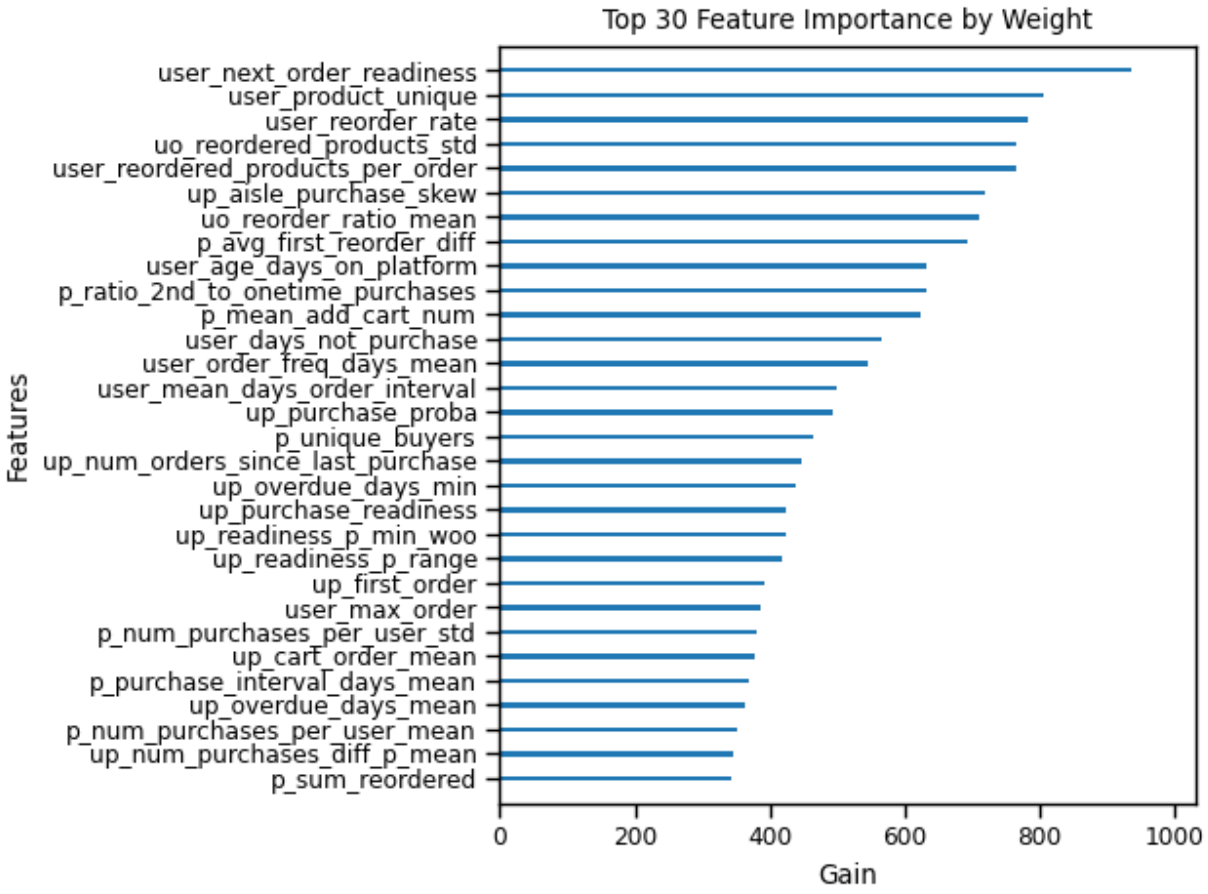
- "user_" is the prefix for user-level features. Similarly, "p_" and "up_" are the prefix for product and user x product level features.
- The suffix "_n5" means the most recent 5 orders
- The suffix "_q" means percentile and "_q20" means 20 percentile
- "_woo" stands for without outliers



Top Features by Weights

Importance by weights measure the number of times a feature is used to split the data across all trees. The more a feature is used in splitting, the more important this feature is to the model. However, be aware that boolean features and categorical features are generally less used in

splitting trees, compared with numerical features. The reason is the former have much fewer unique values, and therefore fewer chances to be chosen as a split point.



To explain the top 5 features by weight:

1. **user_next_order_readiness** measures how ready users are to make the next order based on the time they place the last order and their historical average purchase interval. For example, If a user places an order two days ago, and he usually places orders every five days, he/she is not that ready to make the next order. The user_next_order_readiness is -3, the subtraction between these two dates.
2. **p_reorder_proba** is the reorder probability for products.
3. up_num_orders_since_last_purchase measures how many orders users have gone without purchasing a certain product. If there have been many orders that customers pass on a product, they might just lose interest. Of course, we also need to factor in the purchase interval of different types of products. For example, fresh fruits might be purchased every order while toilet paper is only purchased every few orders.
4. **up_order_rate_since_first_order** is users' order rate on each product after their first purchase. It reflects users' likeness to a product.
5. **p_avg_reorder_diff** is the average order difference between users' first order and second order on each product.

Top Features by Gain

Importance by gain ranks feature importance based on the average gain across all splits the feature is used in. Gain is essentially the improvement in accuracy brought by a feature to the branches it is on. A feature with large gains is more important for making an accurate prediction. There are the features with the top 5 gains:

1. **up_num_purchases_r5** counts the number of times that users bought a product in the most recent 5 orders.
2. **up_purchase_ratio_r5** is the percentage of recent 5 orders where users bought a certain product. For example, if an user bought a product twice in the most recent 5 orders, the up_purchase_ratio_r5 is 40%
3. **up_order_rate_since_first_order**, see the description above
4. **up_num_orders_since_last_purchase**, see the description above
5. **up_purchase_proba_r5** measures the probability that a user reorders the product after the user first bought the product in the recent 5 orders. It's the "r5" version of the feature up_purchase_proba.
6. **up_purchase_interval_days_max_r5** measures a user's max purchase interval on each product based on the recent 5 orders.

Top Features by Cover

Importance based on cover ranks features based on the average number of instances each feature impacts across all splits. Usually, features which split at the top of trees are more likely to affect more instances. Moreover, features used to split more often tend to impact more instances.

1. **up_num_purchases_n5**: see above description
2. **p_num_purchases_per_user_q20** captures each product's popularity by users' purchase time. For each product, it measures the 20th percentiles of the number of times different users have bought it.
3. **p_is_gluten_free** indicates if a product is gluten-free.
4. **up_organic_substitute_num_purchases_n5** captures users' purchases on the organic substitute products. For example, organic strawberries are an organic alternative to regular strawberries. Users who purchase the organic version are not very likely to purchase the non-organic version in the same order. This feature counts the number of times that users purchase a product's organic alternatives in the recent 5 orders.
5. **up_purchase_ratio_n5**: see above description

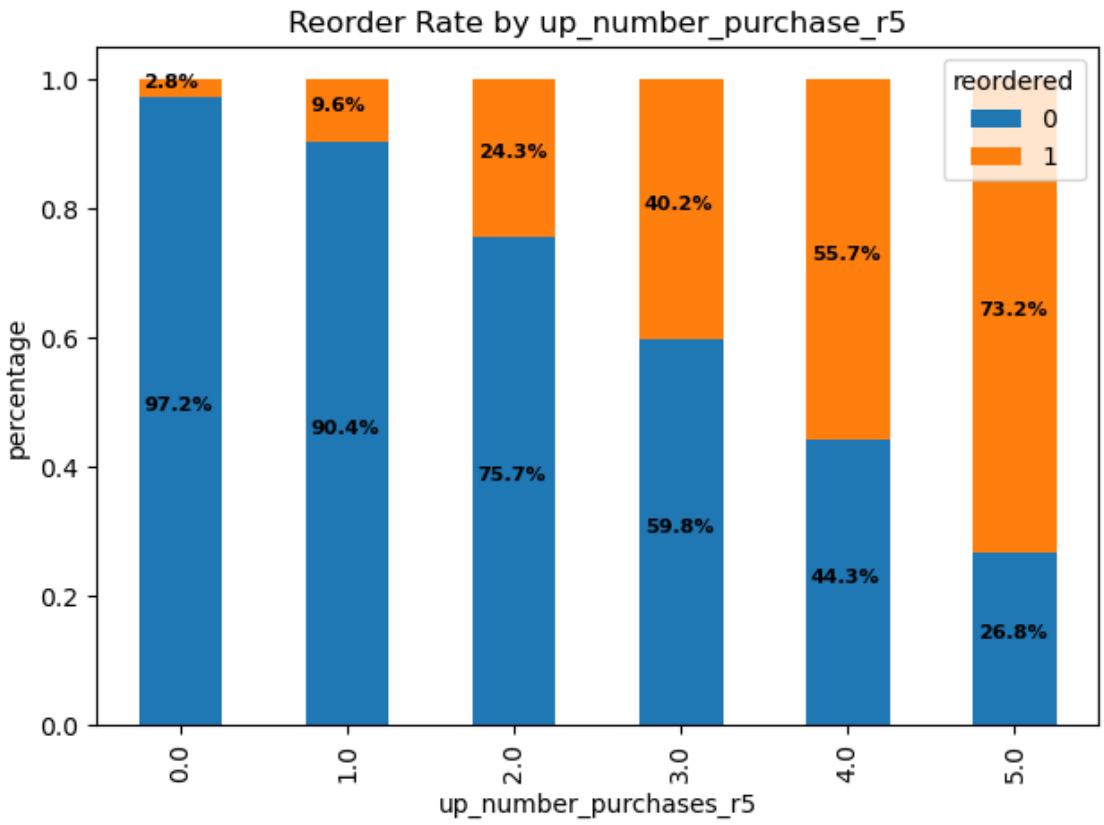
From the three figures above, we know that user x product level features are generally more important than the other two groups of features. Moreover, among user x product level features, the features that capture users' more recent behavior have higher importance.

Insights

Here are some insights I found when doing exploratory data analysis and analyzing the feature importances.

Finding 1: Number of Purchases in Recent 5 Orders

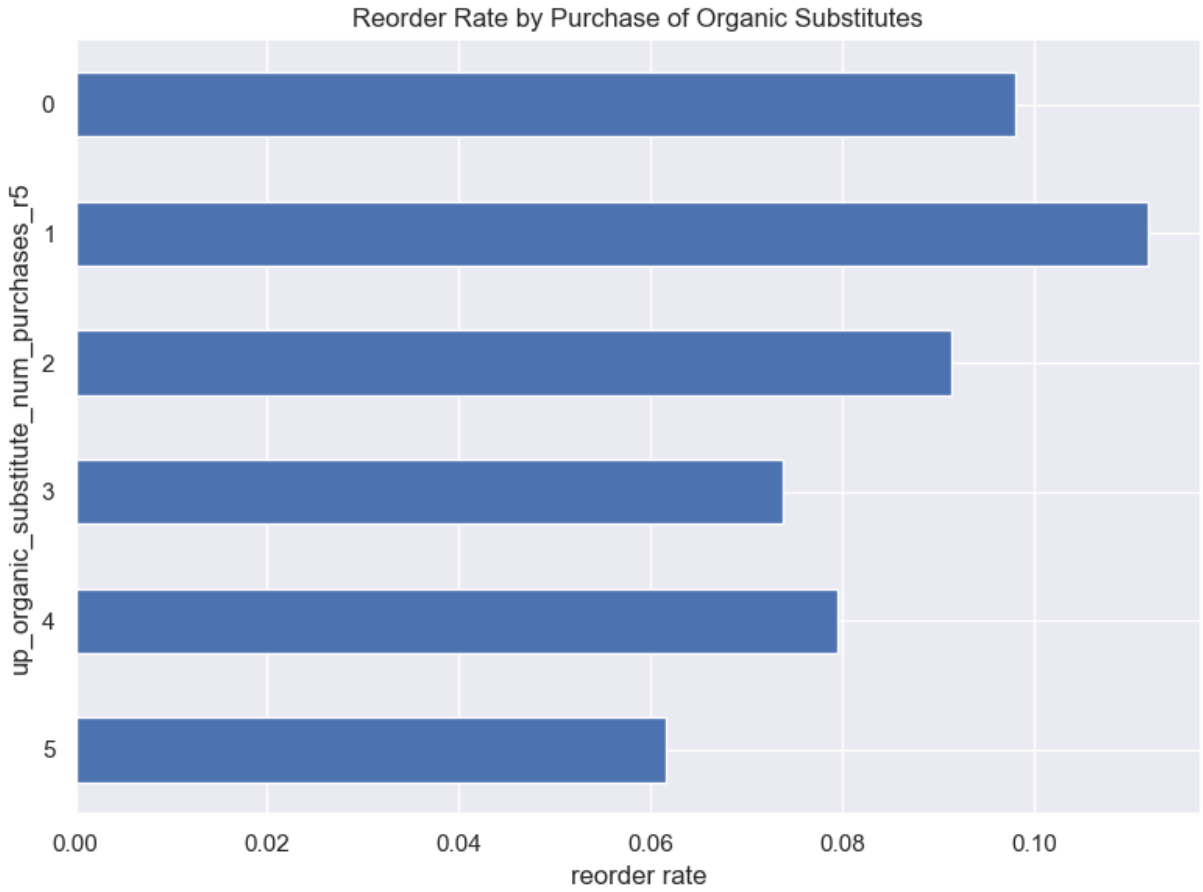
As discussed, up_number_purchases_r5 counts the number of times that each user bought a product in his/her most recent 5 orders. I plot this feature against the reorder rate. We can find that if users bought a product more often in the recent 5 orders, they are more likely to reorder this product in the next order. More specifically, if users didn't buy a product in the near 5 orders, there is only a 2.8% chance that they will reorder it in the next order. Contrarily, if users kept reordering a product in the recent 5 orders, the chances that they will reorder it again in the next order is as high as 73.2%. This finding also aligns with our common sense that the more we have reordered it, the more likely we might reorder it again.



Finding 2: Organic Substitutes

Organic foods have been gaining popularity as more people have become aware of their benefits -- no chemical pesticides, fertilizers, growth hormones, or antibiotic residues. On Instacart, many foods have their generic version and organic substitutes, such as strawberries and organic strawberries, eggs and organic eggs.

For users who buy both the non-organic and organic versions of a product, I want to analyze if the purchases of one version will impact the purchases of the other. From the figure below, we can see that once a user bought a product's organic substitute more than 2 times in the recent 5 orders, the reorder rate of this product will drop significantly.

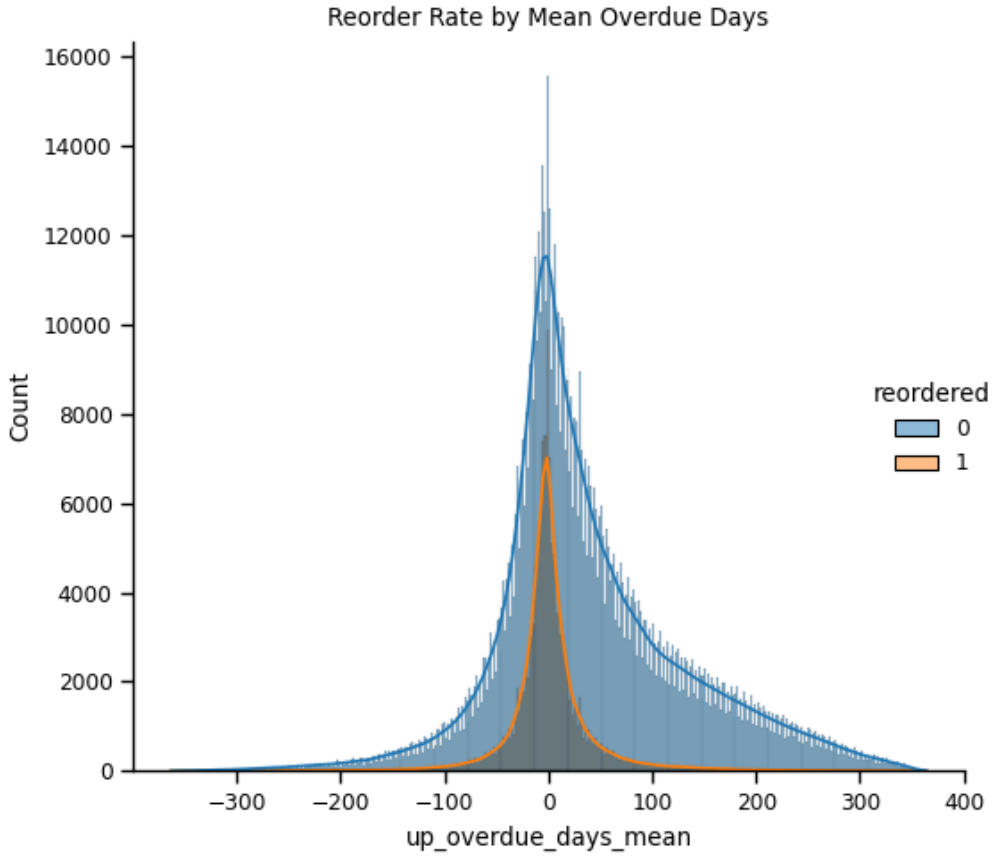


Finding 3: Reorder Rate by Overdue Days

People tend to purchase products with their own cycles. Some customers might purchase fresh vegetables daily while others might just order them weekly. To incorporate this cycle into prediction, I created a feature up_overdue_days_mean to check if it's the right time for a user to reorder a product. Specifically, it's calculated by subtracting users' mean purchase interval on a

product from the number of days they haven't ordered it. A value greater than 0 means the item is overdue for this customer while a value of significantly less than 0 means this product might be early for the customer to reorder.

To understand how the feature, up_overdue_days_mean, facilitates prediction, I've drawn its distribution for reordered and non-ordered products. The distribution of non-reordered products is right skewed. It means that if a product is overdue for too long, the chances of reordering is slim. It has much heavier tails than the distribution of reordered products. This indicates that users usually reorder close to due days. Moreover, if an item is either overdue for a long time or too early for customers to restock, customers are not likely to reorder. This feature is helpful for prediction, but it alone certainly is not powerful enough to tell if reordering or not as these two distributions are overlapping.

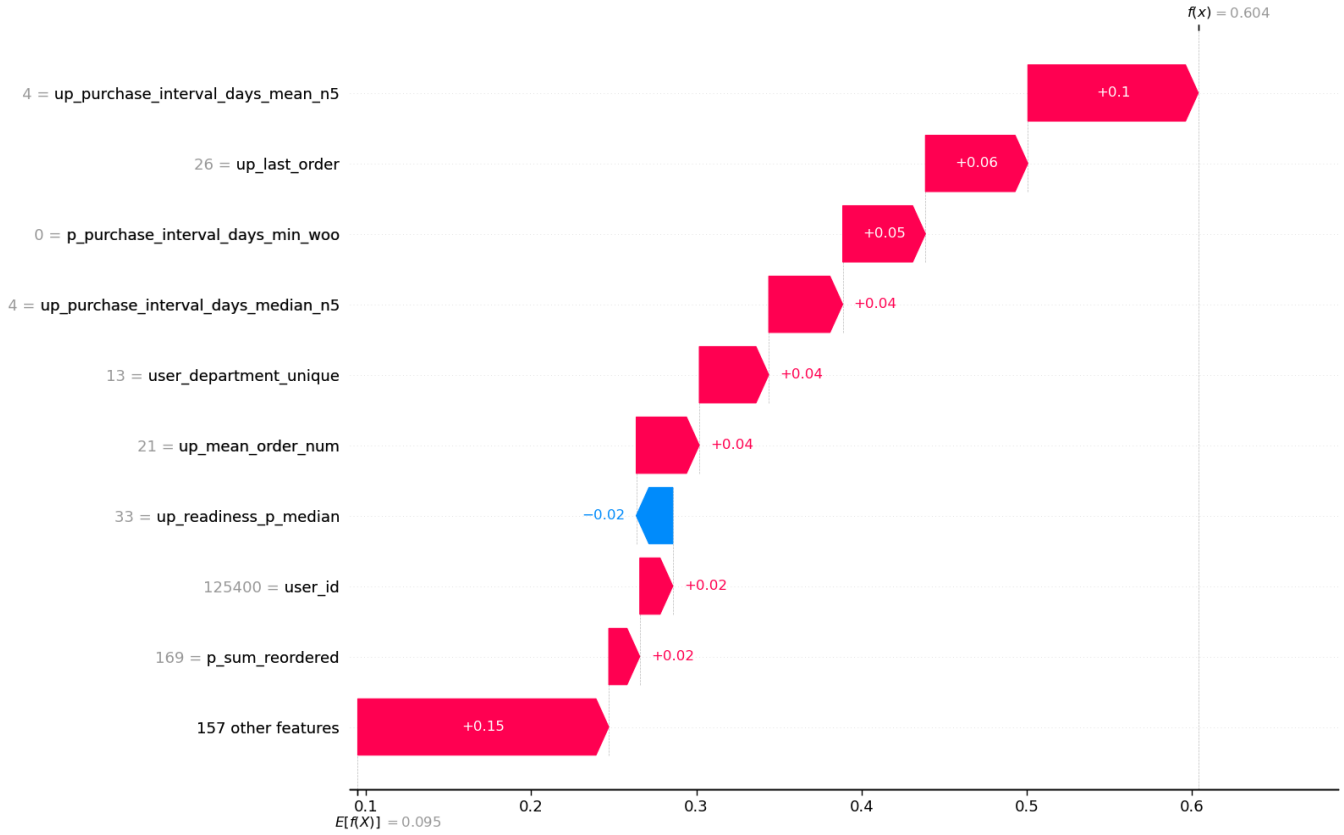


Model Interpretability

Feature importances provided by XGboost give us insights into how the models utilize data, however, it's still not so clear how a model predicts each instance. To solve this issue, I used SHAP values to show the contribution of each feature to the prediction of the model, given any

instances. SHAP values (an acronym from SHapley Additive exPlanations) is a method based on cooperative game theory and it's widely used to increase model transparency and interpretability.

Below is a waterfall plot for a user (uid 125400) and the product milk (product_id 39539). The x-axis is the target value, namely the predicted probability for this user to buy milk in the next order. The base reorder probability, E[f(x)] is 9.5%. The SHAP value for each feature is given by the length of the bar. The value shows how each feature impacts the predicted probability. For example, the feature up_purchase_interval_days_mean_n5 of 4 boosted the probability of reordering by 10%, however, a value of 33 in feature up_readniness_p_meidan reduced the reordering probability by 2%. Combining the impacts of all the features, the final predicted probability of reordering for this user on milk is increased to 60.4%, which is a positive case.



Challenge and Highlights

This is a fun and challenging project to take on. I've learned a lot from tackling the challenges in this project. I want to summarize some of them and my approaches as my takeaways.

Large data set and limited memory

The training set with all the features were up to 14 GB. It's difficult to train such a large data set on my local machine (8GB memory). I tried these methods to ease the memory need:
Optimize data types: This includes converting some string values as categorical and using smaller data types for numerical data based on their range of values. String takes up much more space than categories. By default, Pandas as default stores the integer values as int64 and float values as float64, but some integer features can be well presented in an 8-bit or 16-bit format. I wrote a script to check data ranges and downcast some data types. As the number of bits required to store the data has reduced, memory usage also comes down. This step alone resulted in an 87% reduction in memory usage.
Feature Selection: Removing unimportant features also helps me reduce the memory need significantly. I kept the top 70% feature based on the feature importance plots
GPU support: The GPU support provided by XGboost allows tree training and prediction to be accelerated with CUDA-capable GPUs. The training is much faster and uses considerably less memory.

The need for quick experimentation

Experimentation is the foundation of machine learning and artificial intelligence model development. The faster we can run the experimentation loop -- idea, implement and evaluate, the better results we can achieve in a limited time. This is how I keep the experimentation loop efficient.

- Idea Stage

There are many features I want to try out while doing exploratory and error analysis. However it's impractical to implement every feature given the limited time. Not to mention that it would harm productivity if I get distracted by all kinds of new ideas easily. Instead, I jot down all the ideas and then prioritize them based on my estimated ROI (Return on Investment). For each idea, I estimate its impact on performance (return) and the time it takes to implement (investment).
For example, I once wanted to merge orders placed by the same user on the same day into one order. In the end, I dropped this idea as I found orders that meet the criteria (the same day by the same user) take up only 2% of the total orders. It won't impact the existing feature distribution much even if I implement it.

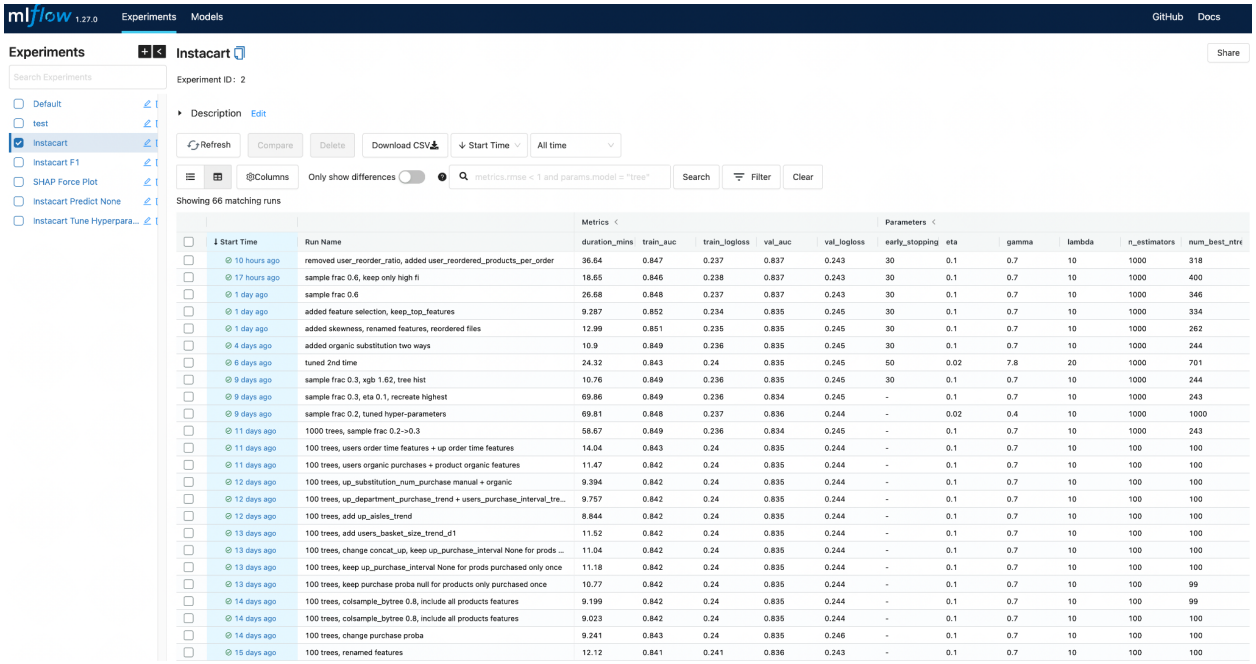
- Implement Stage

Once I decide what to try out, it comes down to the real job of coding. Some code takes a long time to run due to the data size. Therefore I always dry run code with no or very little data to

ensure it does what I expect before feeding all data. Moreover, I also try to avoid duplicate computation by caching my results often. Furthermore, I always prefer vectorization over loops whenever possible. For the areas where I have to use loops, I try to use multiprocessing to accelerate the process. This way, I keep the code writing and validation process as efficient as possible.

- Evaluate Stage

I used MLflow to manage my experiments. MLflow is an open-source platform to manage the ML lifecycle, including experimentation, reproducibility, deployment, and a central model registry. With MLflow, I can create different experiments and under each experiment, I can have many runs. In each run, the performance metrics and model parameters are logged for comparison and reproducibility. The figure below is a example of my experiment “Instacart”, where I have tried different runs to optimize the model



Local Threshold

At first, I used a global/universal threshold to convert predicted reorder probabilities into binary results. The global threshold is found by optimizing the F1 score in the training set. Every user and product pair with a predicted probability greater than this global threshold is predicted as reorder, and vice versa. With this approach, my Kaggle Leaderboard performance is around 0.39 (F1). While I was stuck at this performance, I found some discussions online about the local thresholding, which is to a local threshold for each user to maximize the F1 expectation. I began to read the papers shared in the discussions and then transferred to the local approach

instead. Please see the Thresholding part for more details. This change in the threshold method boosted my LB performance by 0.1.

Future Work

More Substitute Products

From Finding 2, we've seen that purchases of organic and non-organic alternatives impact the reorder rate of the counterparts significantly. However, organic and non-organic are only one kind of substitution. Many other types are present. Some are brand competitors, such as Pepsi and coca cola. Others might just be similar products in different packages, such as bags of bananas and bananas. If we can label all the substitutes, I believe the model accuracy can improve further.

Utilize More Data

Even though I tried my best to reduce my memory needs, I can only use 60% of my training data on my local computer. If more computation resources are available (using a more powerful computer or cloud computing service), the model performance could be further improved with more data. Also, we can expand the training set to be even larger by pretending that one of the previous orders of a user is the last order. For example, we can treat users' first to third to last orders as a training set while the second to last order as a test set to predict.