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.

At first sight, this is a multi-class classification problem where each previously bought product is a class. However, this would be very complicated as we might end up having too many classes for highly active buyers. Instead, I approach it as a binary classification problem. For each user and each product that a customer previously bought, I trained an XGBoost Classifier to predict its reorder probability in the next order. Then, I feed the predicted probability into a script that maximizes the expected F1 score and outputs products that are most likely to be reordered for each user. My final prediction on the test set reached an F1 score of 0.404, roughly at the 240th position in this competition.

I put significant efforts into feature engineering. I designed three types of features – user-level features, product-level features, and user-product interaction features – to capture the characteristics of each entity. User-level features capture users' profiles, such as account age, user product preference, order frequency, user order time and behavior trend related features. Product-level features measure product characteristics, such as product popularity (sales), reorder rate, general product purchase cycle, etc. User-product interaction features capture users' behavior on each product. They are also the most important feature group for this task. They include users' purchase times on each product, reorder rate, purchase interval, order interval, if users are in cycle for reordering, and their purchases on substitute products, etc. For more details, please refer to the Feature Engineering section and the code repository.

After feature engineering, I have 242 features in total. It's known that too many features tend to slow down the training and may not generalize well. Therefore, I first dropped highly correlated features and then used recursive feature elimination with cross validation (RFECV) to select the most important features. RFECV recursively removes the least important feature from the model and uses cross validation to evaluate the model performance under each new set of features. In the end, I was able to reduce to 123 features with a moderate improvement on performance (+0.003 on F1 score).

During the modeling stage, I used MLflow to track model performance and log parameters. This has ensured model reproducibility and traceback. After the model was built, I used an explainable

AI tool, SHAP, to explain the outputs of the XGBoost Classifier. We can calculate SHAP values for each individual prediction and know how the features contribute to that single prediction. 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 bought, 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.

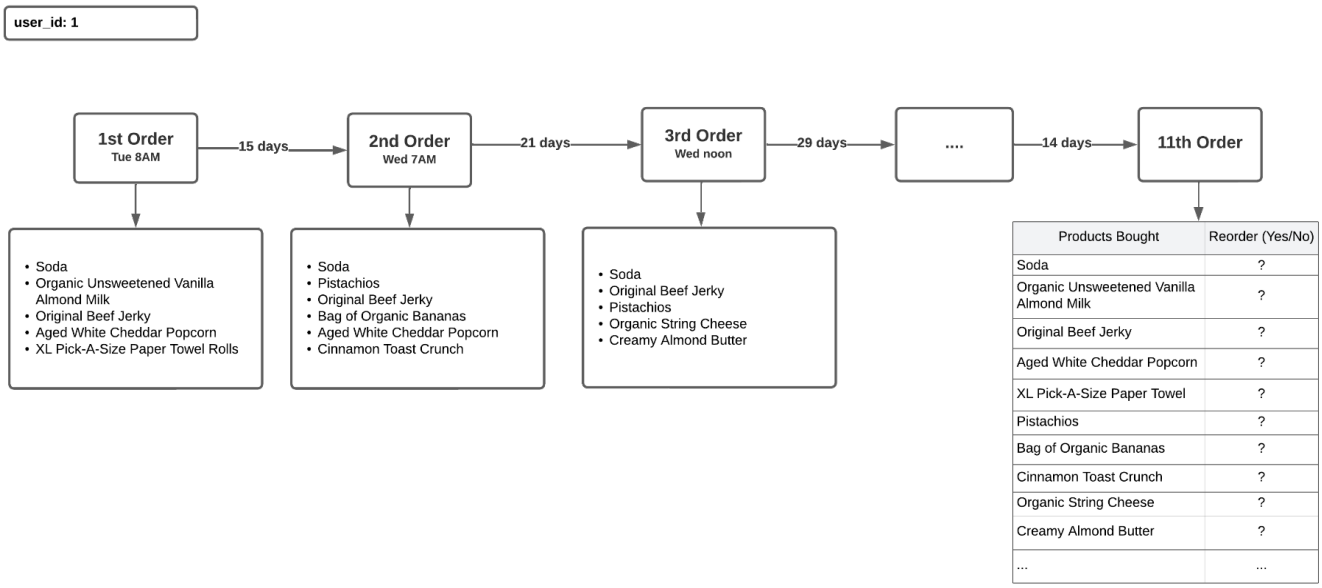

Figure 1: User Order History and Prediction Task

Exploratory Data Analysis

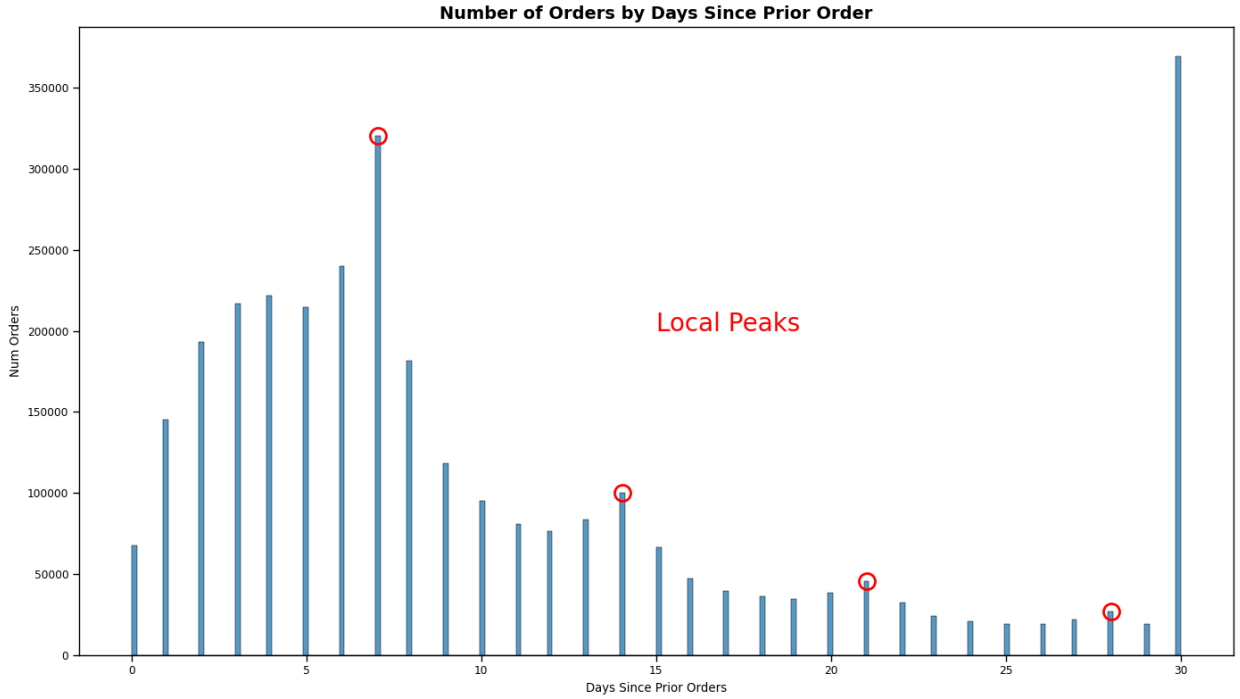

Figure 2: Number of Orders by Days Since Prior Order

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

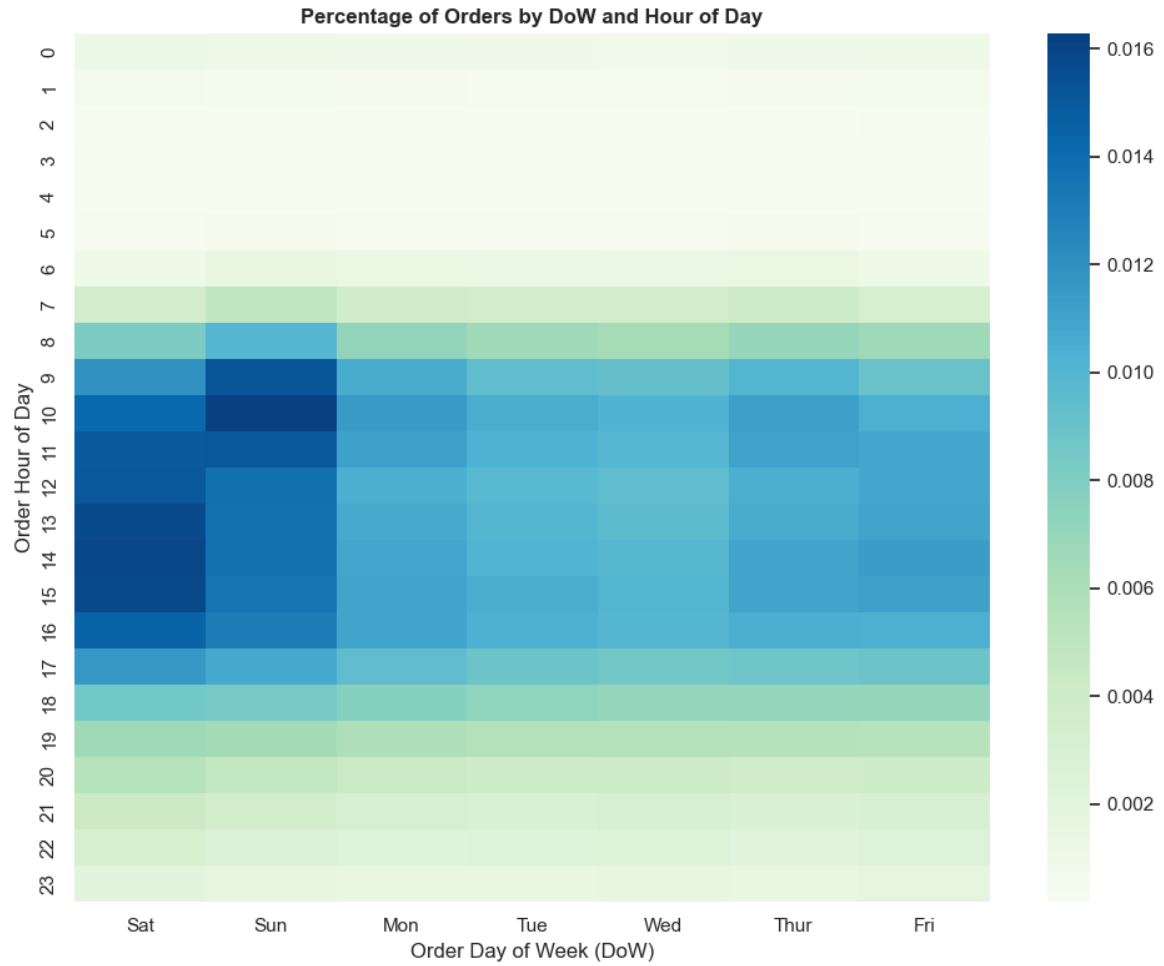

Figure 3: Order Distribution by Day of Week (DoW) and Hour of Day

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

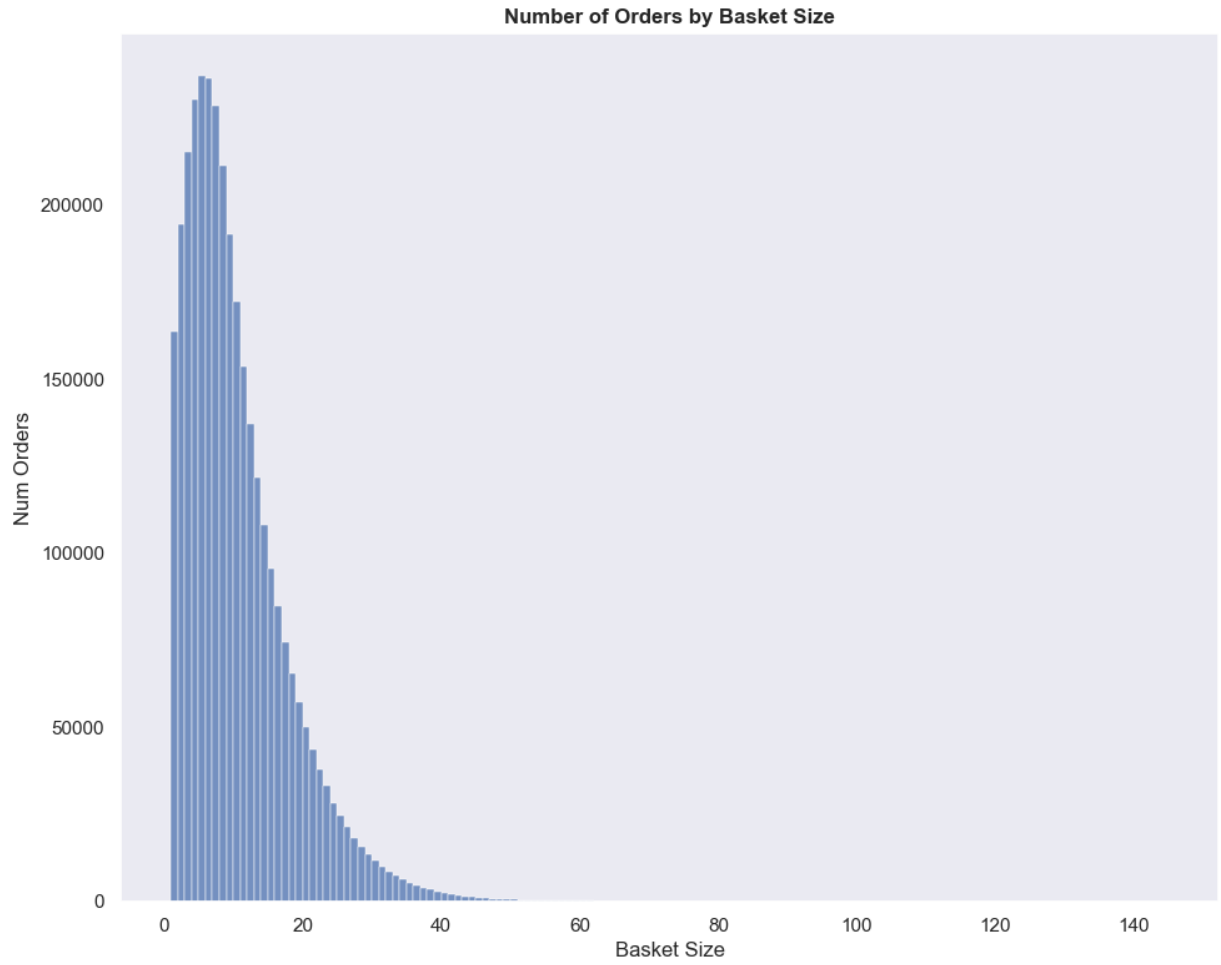

Figure 4: Order Distribution by Number of Items in Basket

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.

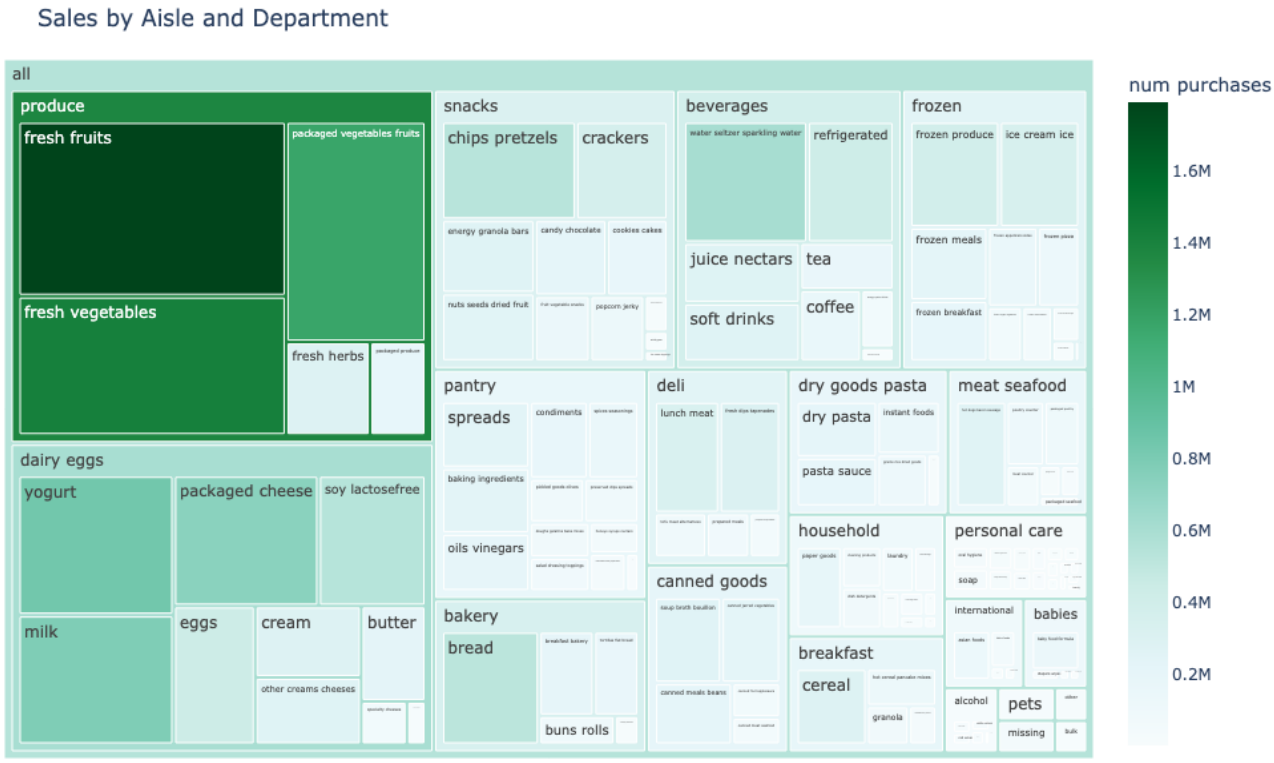

Figure 5: Sales by Aisle and Department

The produce department has the most sales and then follows the dairy eggs department. In Produce department, fresh fruits, fresh vegetables, and packaged vegetable fruits are the most ordered items. On the contrary, pets, babies departments have least sales.

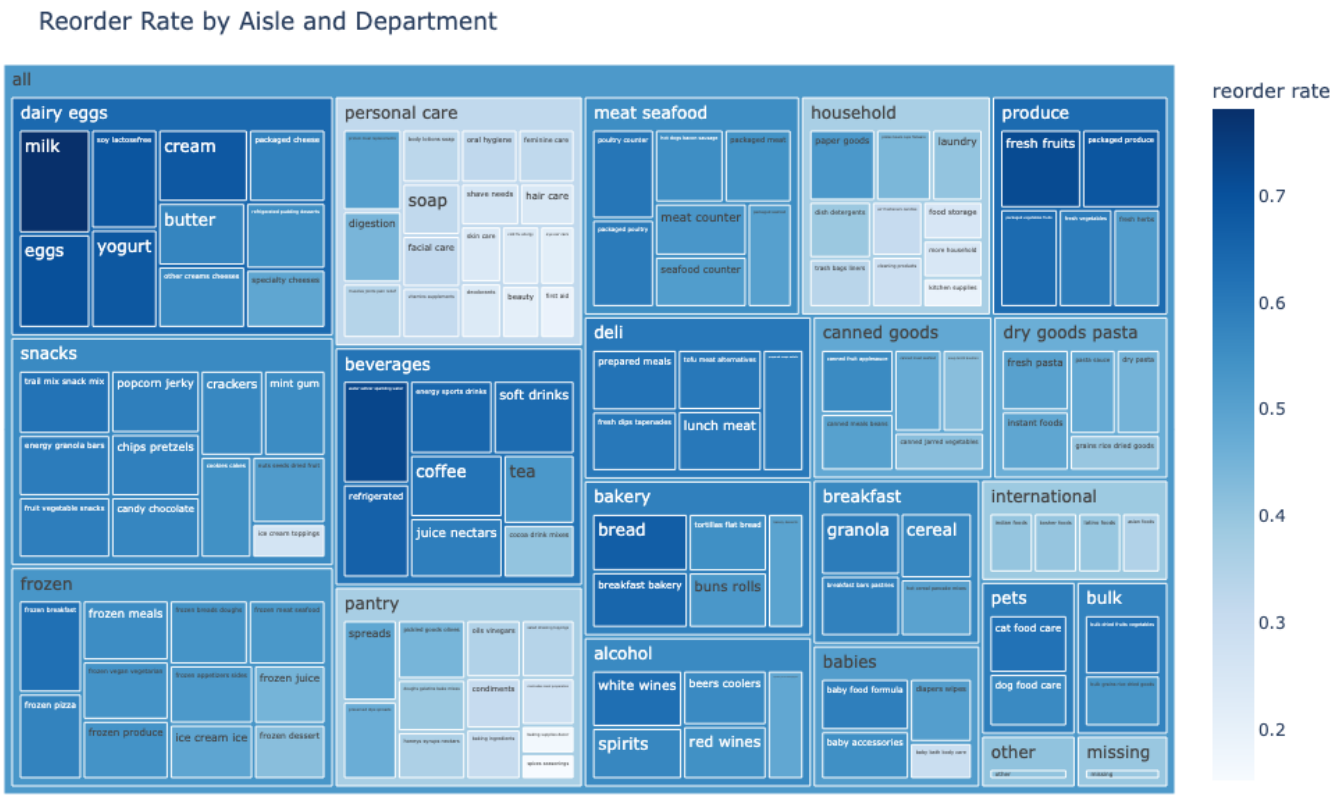

Figure 6: Top Reordered Department Aisles

Just like any other grocery stores, dairy eggs, beverages, and produce are the most reordered departments. Personal care and household are the least reordered departments. We can also see the top reordered aisles in each department from the chart.

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 user-level features in order to answer these questions:

- How long has the user been with us/Account age and number of orders on the platform?
- How often do they place an order? and when do they usually place orders (a particular day of week or a particular part of day)?

- What's the basket size for the user? How diverse is their basket?
- Do they have any special product preference, such as organic, gluten-free, and Asian food?
- 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 are they to place the next order based on their average purchase intervals?
- Is there any trend in users' behavior, such as placing orders more and more often or having a larger and larger basket size?
- Users segment features based on the KNN algorithm

Product Features

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

- How popular each product is, aka, how many times they are purchased and by how many different users?
- How are they added to the shopping cart? Often added first or added very late?
- Products reorder rate, ratio of one-time purchases and second-time purchases and when users make their first reorder?
- How often products are reordered? Every week or every month? Different types of products (daily essentials versus pantry) can have very different purchase interval.
- How can we use vectors or natural language processing to represent these products?

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. When measuring cycles, I use two units. One is in terms of days and the other is in terms of orders. User x product features intent to answer these questions:

- How many times does a user buy a product and when do they buy it?
- How often (in days & orders) do users buy and reorder each product? Every week or every month? every order, or every other order?
- What's users' reorder tendency on a product? Reorder tendency is defined as the historical reorder probability of a user on a product after the first trial.
- How long have users gone without this product (in days and orders)?
- How ready are users to make the next purchase on this product, given users' own purchase cycles?
- How ready are users to make the next purchase on this product, given the general purchase cycle on this product?
- Any changes in user's behavior? Do they buy this product more and more often? Do they start to buy more/fewer products in the same aisle and department.
- How are users purchasing the substitute products? Substitute product is defined as a product that is bought in place of the other, such as Diet coke and coke Zero, organic apples and regular apples.

It makes sense that users' recent orders are more representative of user behavior, 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 have the same meaning as the non “recent 5” features. The only difference is they are calculated based on only the recent 5 orders of users rather than the entire history orders.

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 efficient algorithm to speed up the experimentation cycle.
I used the XGB classifier to predict the probability of reordering for each user and product. In other words, the entity of the training set is each pair of user id and product id. I used log loss as the optimization metric during model training.

F1 Maximization

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 outcome. 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 0.5. However, my experiment shows that using different thresholds for different users can improve F1 score further than using the same threshold for all users. To illustrate this, let’s use two examples below:

P(A)	0.4
P(B)	0.3

		F1 in Different 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 7: 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 to be 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 8: 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 must choose different thresholds for different users to get the max expected F1 score for the whole dataset.

Performance

Due to the limitation of computer memory, only 40% 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 9: Test Performance

Besides validation performance, the prediction on the given test set has achieved an F1 score of 0.403 on the Kaggle Leaderboard.

Feature Importance

XGBoost provides three ways of measuring feature importance, weights, gain and cover. I've analyzed the top 5 features under each measurement.
For better readability, note below name conventions:

- "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 "_r5" means the most recent 5 orders
- The suffix "_q" means percentile and therefore "_q20" means 20 percentiles
- "_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.

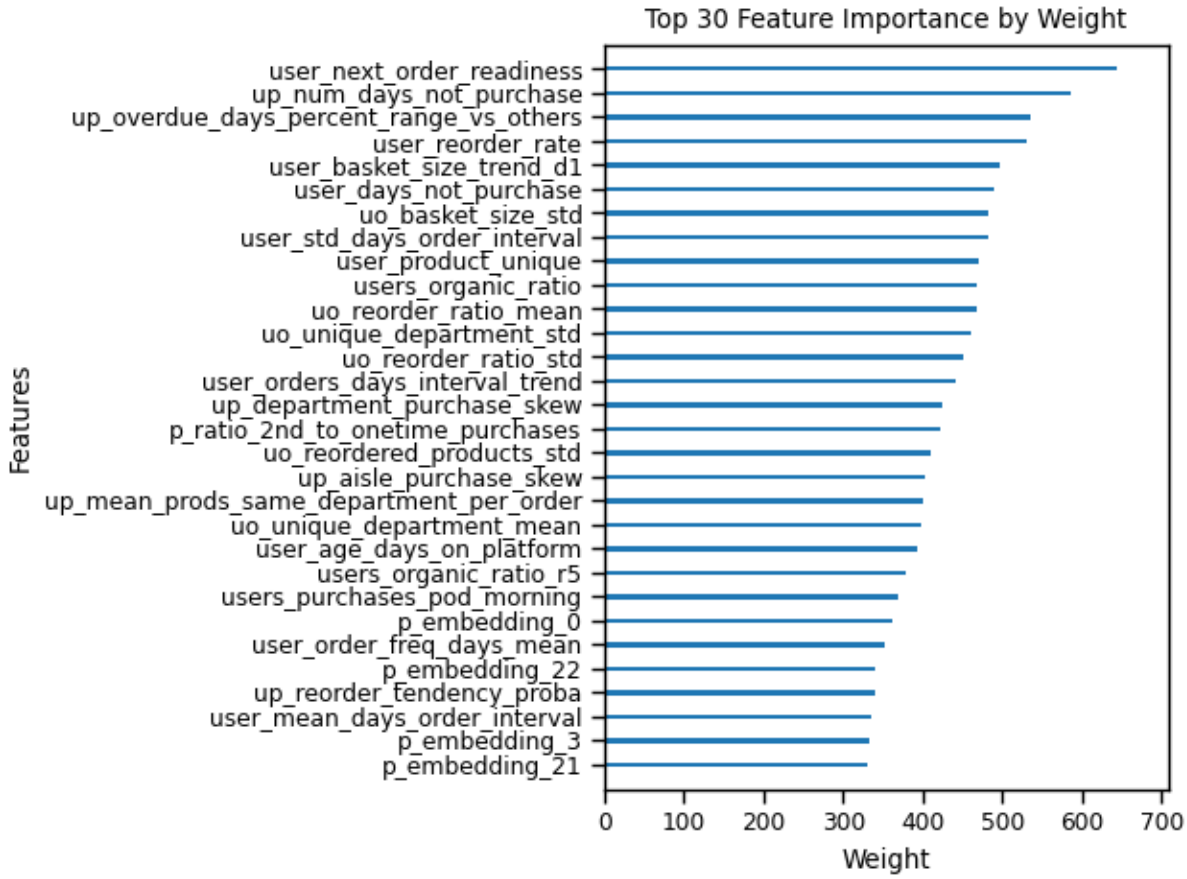

Figure 10: Top 30 Features by Weight

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 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 in this case is -3, the subtraction between these two numbers.
2. **up_num_days_not_purchase** measures how many days that users haven't purchased a product.
3. **up_overdue_days_percent_range_vs_others** is a percentage that uses number of days since users last purchase the product divided by the range of product purchase interval of other users. For example, user A hasn't bought coke cola for 10 days. By analyzing all the users who bought coke cola, we know that the min (outliers excluded) purchase interval (days

difference between purchasing a coke cola) is 2 days and max purchase interval (outliers excluded) is 15 days. Then this feature up_overdue_days_percent_range_vs_others is 10/13 for user A. A value significantly larger than 1 means a user hasn't purchased a product longer than the general population, and therefore the reordering probability will be low.

4. **user_reorder_rate** measure the percentage of products that users bought are reorders
5. **user_basket_size_trend_d1** measures the trend of users' basket size changes. The trend is defined as the coefficient of linear regression of a user's basket sizes.

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 split on. A feature with large gains is more important for making an accurate prediction. There are the features with the top gains:

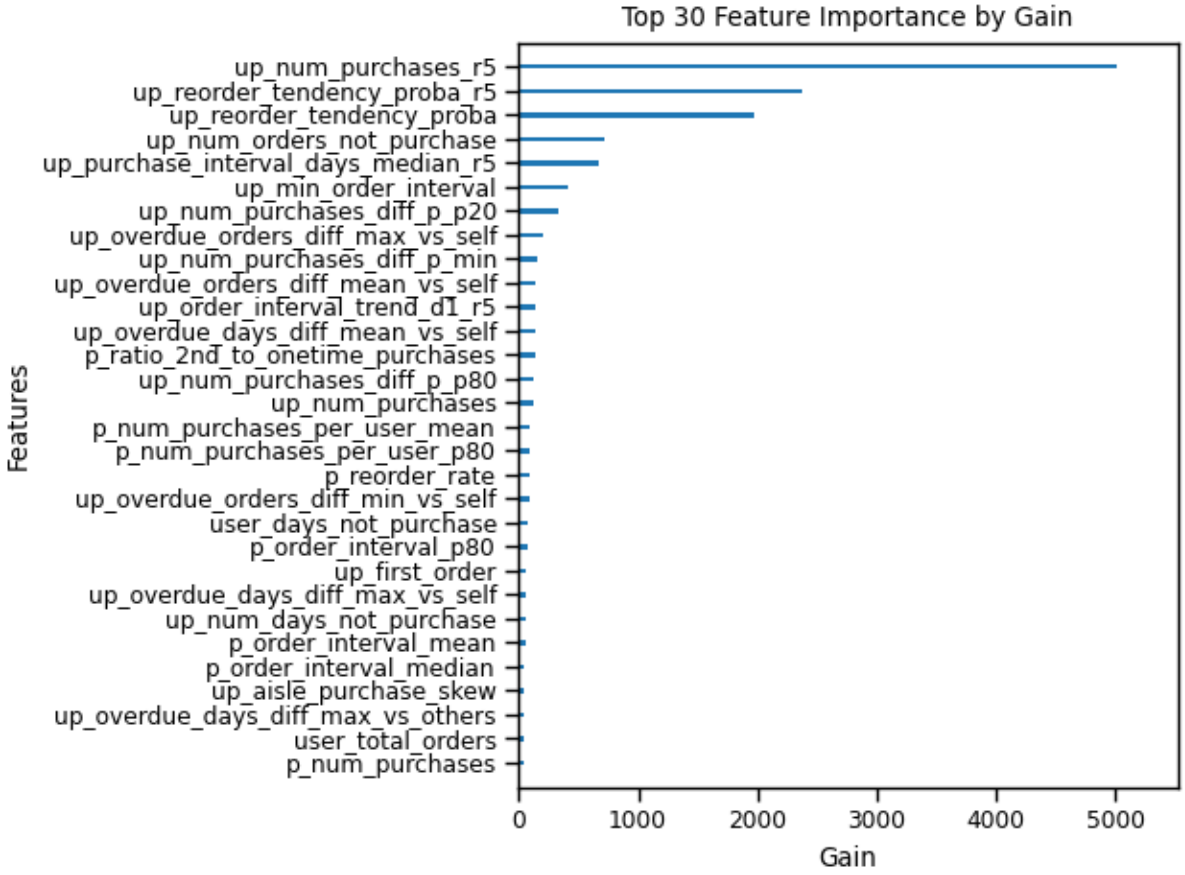

Figure 11: Top 30 Features by Gain

1. **up_num_purchases_r5** counts the number of times that users bought a product in the most recent 5 orders.
2. **up_reorder_tendency_proba_r5** is the historical reorder probability for a user and product based on the recent 5 orders.

3. **up_reorder_tendency_proba** is the historical reorder probability based on all orders of a user.
4. **up_num_orders_not_purchase** counts the number of orders past since users last bought a product.
5. **up_purchase_interval_days_median_r5** is the median days differences between two purchases of a product for a user based the recent 5 orders.
6. **up_overdue_orders_diff_min_vs_others** measures how many orders a product is overdue for a user compared with the min order interval of other users. For example, a product is often bought by other users at least every 2 orders, and a user of interest hasn't bought this order for 4 orders. Then this product is overdue for 2 orders for this user.

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.

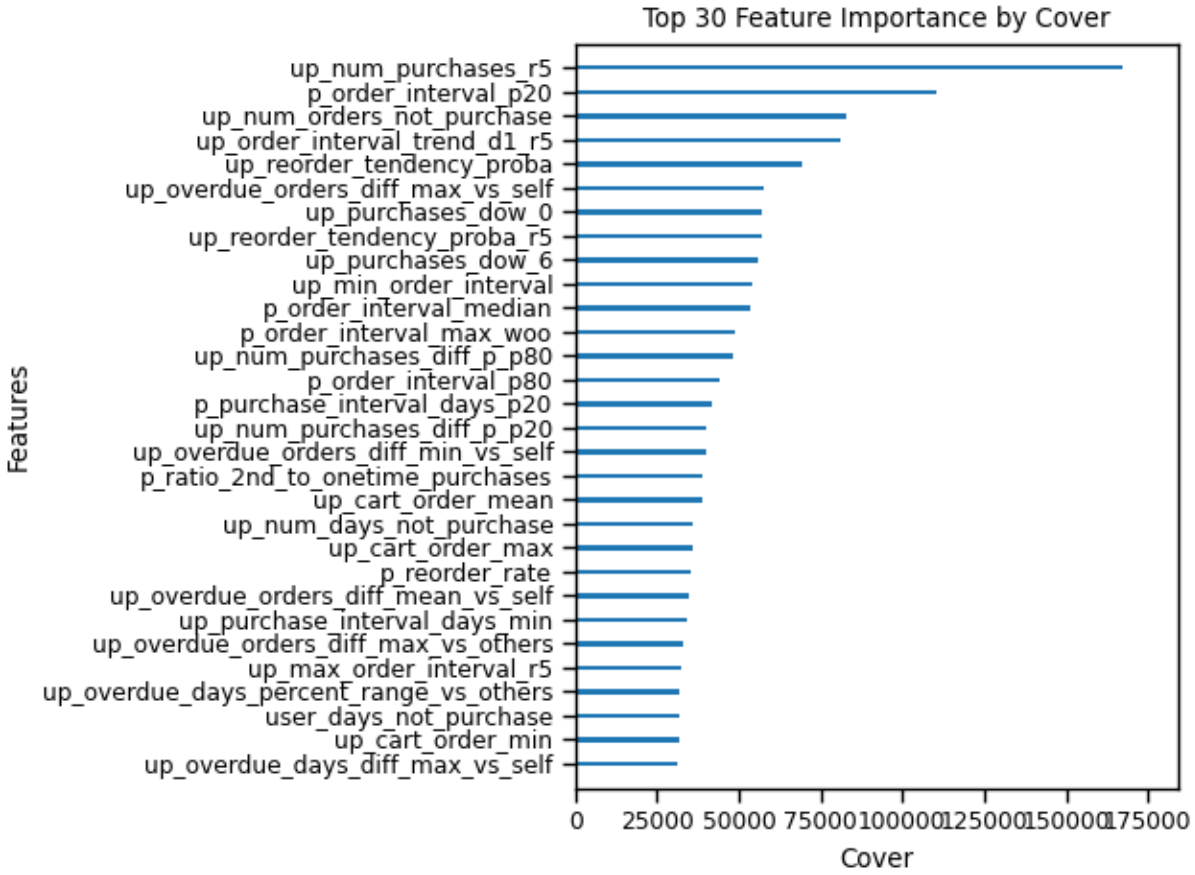

Figure 12: Top 30 Features by Cover

1. **up_num_purchases_r5**: see description above

2. **p_order_interval_p20**: the 20 percentiles of the order interval (order number difference) for a product
3. **up_num_orders_not_purchase:** see description above
4. **up_order_interval_trend_d1_r5** measures the trend of users' order interval on a product based on the recent 5 orders. Trend is represented by the coefficient of a linear regression.
5. **up_reorder_tendency_proba:** see description above

From the three figures above, we know that user x product level features occupy the top of feature importance lists, and therefore they are 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 importance.

Finding 1: Purchases in Recent 5 Orders

As mentioned, 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 for users who bought a product more often in the recent 5 orders, they are more likely to reorder this product in the next order. More specifically, for products that were not ordered in the recent 5 orders, only 2.8% of them were reordered by users in the next order. On the opposite side, for products that were purchased 5 times out of the recent 5 orders, 73.2% of them are reordered again in the next order. Also, the reorder rate keeps increasing as up_number_purchases_r5 increases.

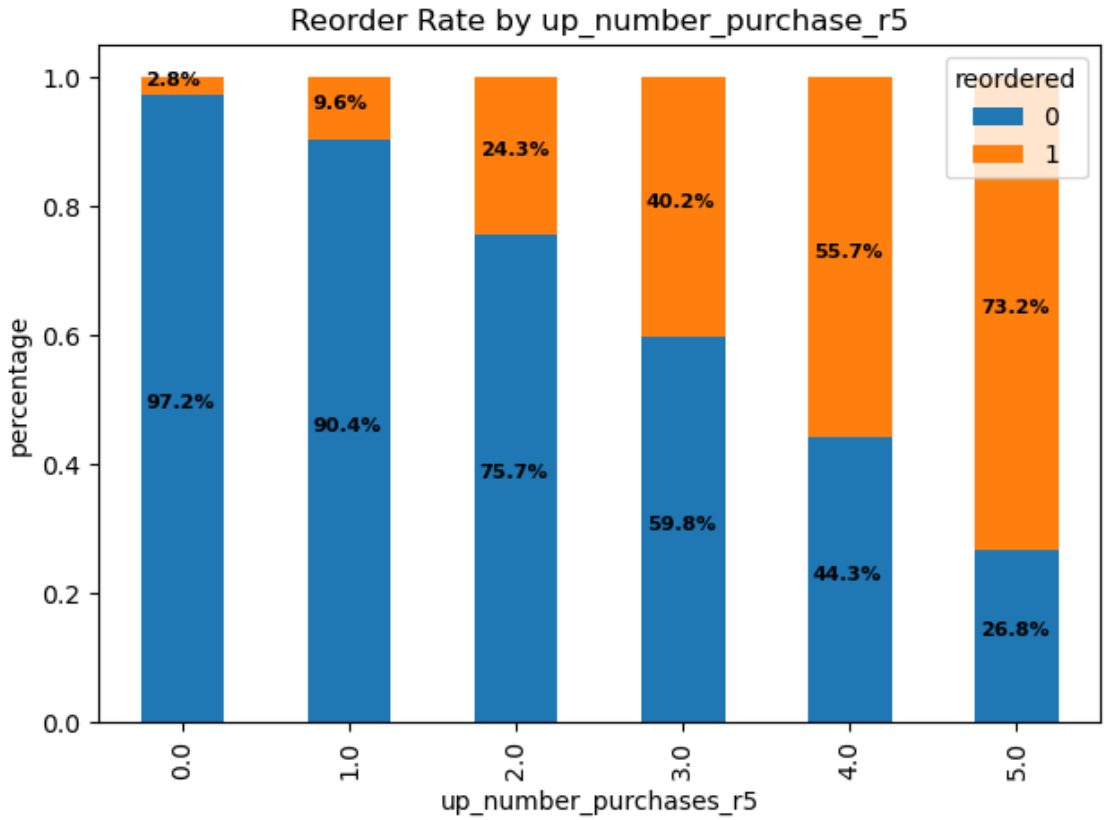

Figure 13: Finding 1 about num of purchases in recent 5 orders

Finding 2: Purchases in Recent 5 Orders and Reorder Tendency

In Finding 1, we have seen the impact of the number of purchases in recent 5 orders on the reorder rate. In the last Feature Importance section, we see that users' reorder tendency is also highly predictive of features. We will see how these two features help us predict if a user will reorder a certain product. Figure 14 is the scatter plot of these two features. As both features are discrete, jittering is added to the plot for readability.

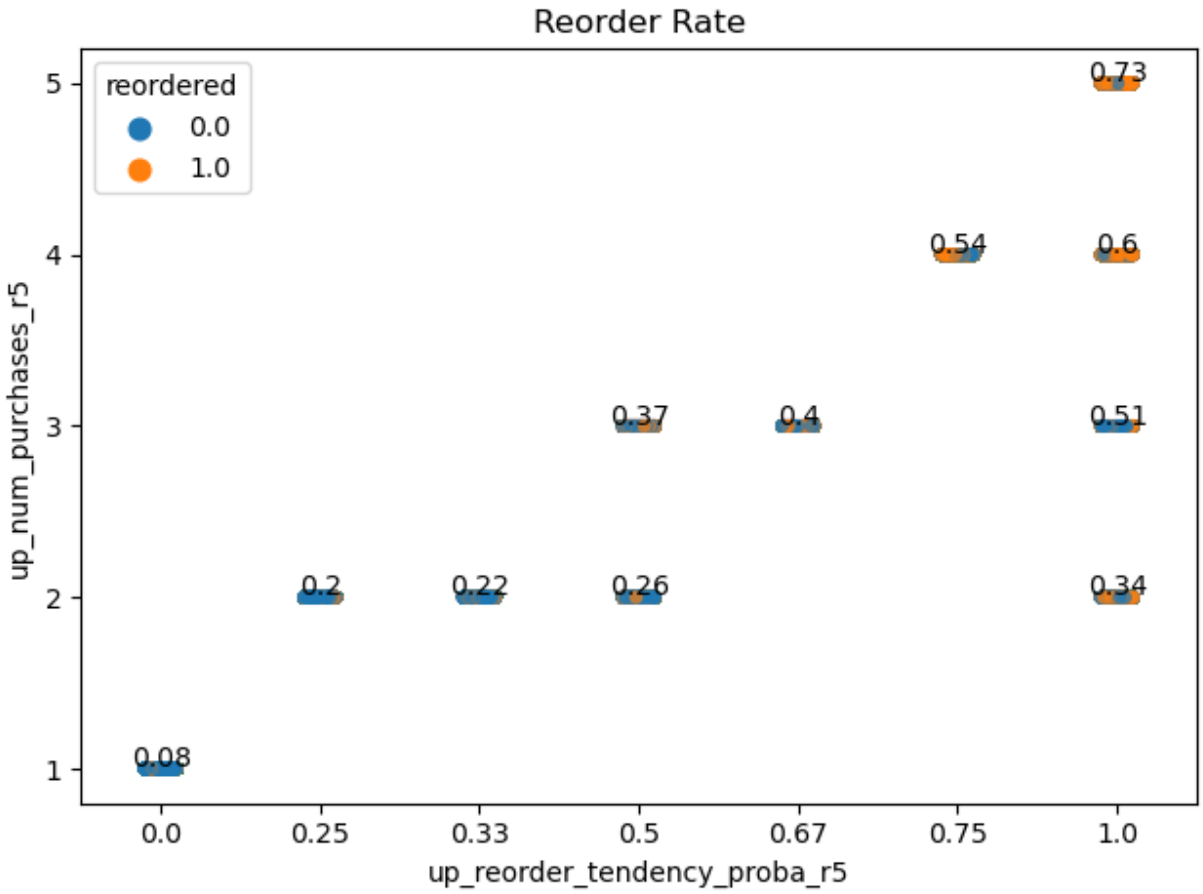

Figure 14: Scatter Plot with Jittering

We can see that, when feature up_number_purchases_r5 equals to 5, up_reorder_tendency_porba_r5 can only be 1, and the reorder rate in this situation is 73%, exactly as we have seen in the finding 1. When up_number_purchases_r5 equals to 4, reorder tendency probability could be 0.75 or 1. Reorder tendency probability is 3/4 when users' first order of a product is at the 5th to the last, and they reorder 3 times out of the next 4 orders. Reorder tendency probability is 1 when users' first order of a product is at the 4th to the last, and they reorder 3 times out of the next 3 orders. The reorder rates on those users and products are 0.54 and 0.6 respectively. As we move down along the y-axis, we can notice that given the same up_number_purchases_r5, reorder rate increases with reorder tendency.

Finding 3: 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 products, I want to analyze if the purchases of the 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 itself will drop significantly.

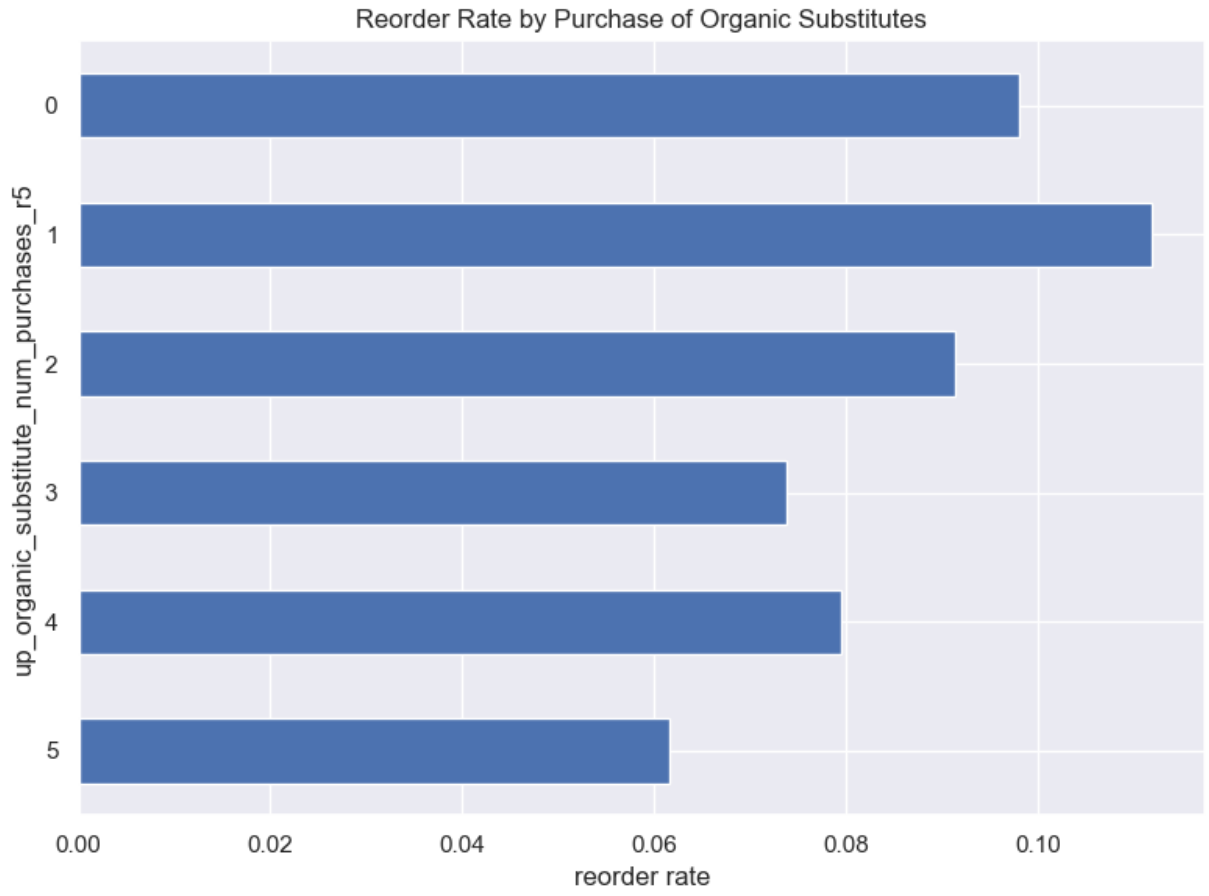

Figure 15: impacts of substitute purchases

Finding 4: 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 past since they last 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. Naturally, a value significantly larger than 0 means users haven't reordered this product for an unusually long time and it might indicate that users have forgot or lost interest on this product.

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 chance of reordering is slim. Ordered distribution are more centered around 0 than non-ordered distribution. This indicates that users usually reorder close to due days. Moreover, non-ordered distribution has much heavier tails. It means that if an item is either overdue for a long time or too early for customers to restock, customers are not likely to reorder. By analyzing the shapes of this chart, we see that 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.

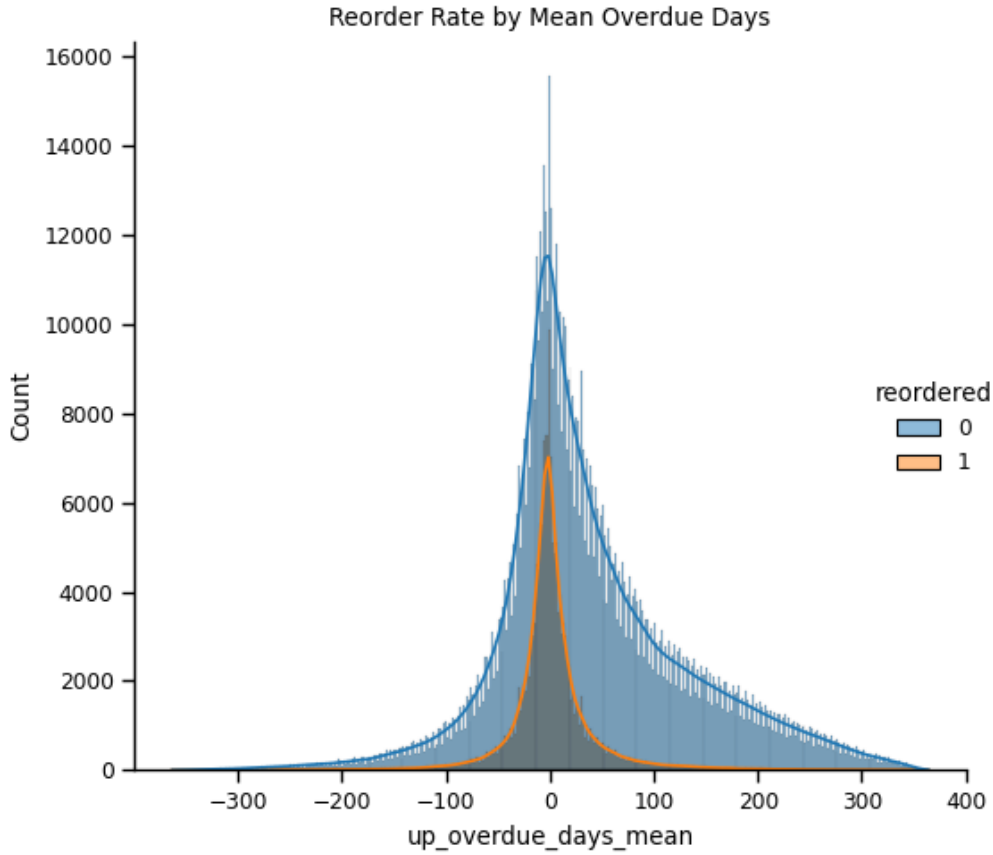

Figure 16: overdue days and reorder

Model Interpretability

Feature importance 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 any instances. SHAP values (an acronym from **SH**apley **A**dditive ex**P**lanations) is a method based on cooperative game theory and it's widely used to increase model transparency and interpretability.

Below is an example of using SHAP values to understand why the model predicts user (uid 125400) will product milk (product_id 39539) in the next order. The contribution of each feature and how it impacts the final prediction are shown in the waterfall plot. 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 9.2%. 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_reorder_tendency_proba value of 0.75 boosted the reordering probability by 7% and the up_num_purchases_r5 being 4 further boosted reorder probability by 5%. however, a value of 15 in feature up_first_order reduced the reordering probability by 1%. Combining the impacts of all the features, the final predicted probability of reordering for this user on milk is increased to 28.3%, which is a positive case.

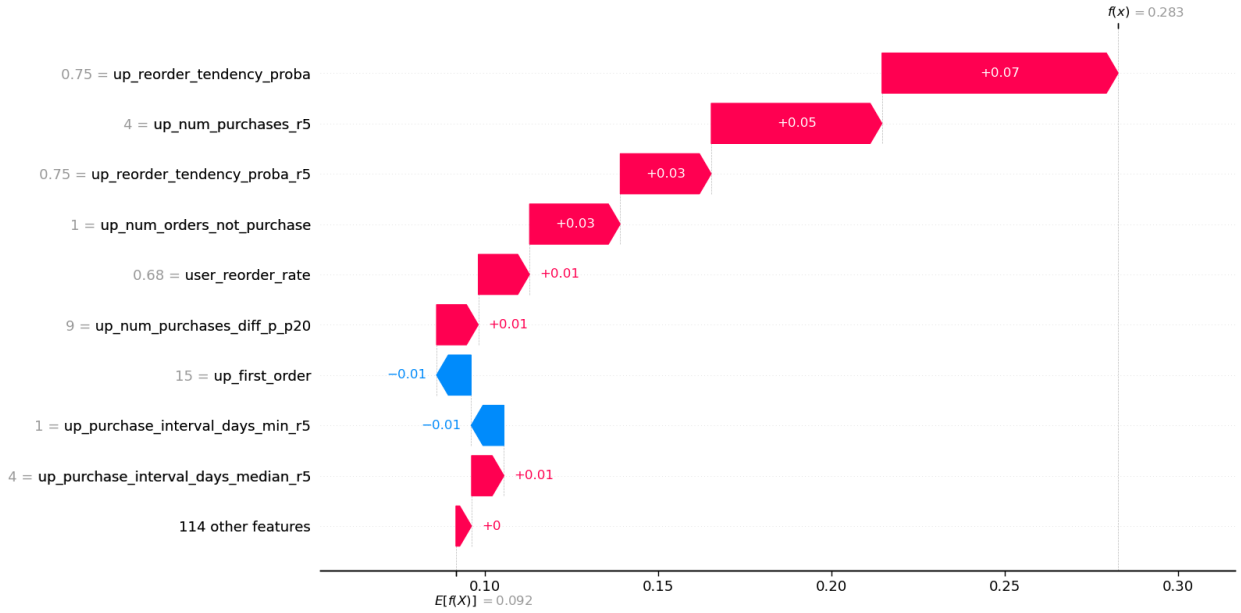

Figure 17: Example of Shap Values

Challenge and Highlights

This is a fun and challenging project. I've learned a lot from it. I want to summarize some of challenges and my approaches/trials as my takeaways.

Large data set and limited memory

The training set with all the features added were up to 14 GB. It's really difficult to train such a large data set on my PC (8GB memory). I tried these methods to ease my 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 first removed highly correlated features (Pearson correlation greater than 0.98) and then used recursive feature elimination to reduce to 123 features.

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 my 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 will harm productivity if I get distracted by various new ideas easily. Instead, I jot down all the ideas once then come to me 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) so that I can check if my time would be well spent.

For example, I once wanted to merge orders placed by the same user on the same day into one order because I believe users forgot some items in their first order. Those orders are intended to be 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. So, I believe 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 first ensure it does what I expect before feeding all data. Moreover, I also try to avoid duplicate computation by saving my results often. Furthermore, I always prefer vectorization over loops whenever possible. For the areas where I must use loops, I try to use multiprocessing to accelerate the process. This way, I keep the coding and validation process as efficient as possible.

- Evaluate Stage

I used MLflow to manage and track 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 later. The figure below is an example of my experiment "Instacart", where I have tried different runs to optimize the model

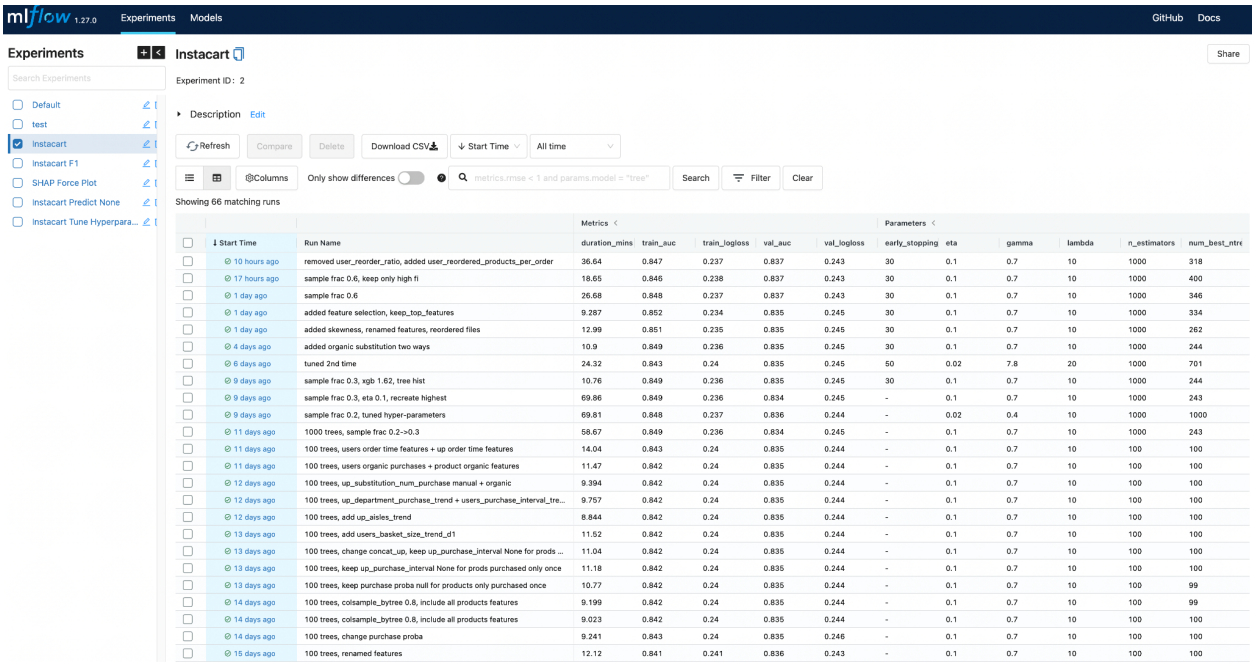

Figure 18: MLFlow Usage Example

Product Embedding

I also tried to capture the relationship between products by using the word2vec algorithm. Word2vec is natural language processing technique that learns word association from a large corpus of text. As the name implies, word2vec can represent each word by a vector and those vectors capture the semantic and syntactic qualities of words. To use this algorithm in this competition, orders can act like sentences and each product id can act like words. I first got a 100-dimension vector for each product and then used PCA to downcast them to 36 dimensions with 80% variance retained. However, feeding these vector features into the model did improve my leaderboard score slightly by 0.01. However, it's hard to interpret what each dimension represent.

The other thing I tried to use the results of this algorithm to find the most "similar" products. Similarity here means the probability that products that could replace with each other in the context (other products in the same order). For example, it can identify that Bags of Banana is the most "similar" product for product Banana in the context of all user orders. This means that Bags of Banana and Banana can act like alternatives/substitutes. I used this method to identify substitute products and calculate the users' purchases on them.

F1 Maximization

At first, I try to optimize F1 score by finding a global threshold that can give largest 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 a local threshold for each user to maximize the F1 expectation. I began to read the papers referred in the discussions and then transferred to the local approach instead. Please see the Thresholding part for more details. This change in the thresholding method boosted my LB performance by 0.1.

Recursive Feature Elimination with Cross Validation (RFECV)

RFECV is a feature selection technique that recursively removes least important features at a certain rate (one by one if you set hyperparameter “step” as 1) and uses cross validation to check the model performance under each set of features. The Scikit-learn RFECV has implemented this feature selection method, but it doesn’t support the early stopping function when you use it on XGboost. However, early stopping is crucial for prevents overfitting in XGBoost. Therefore, I implemented my own RFECV function in python with early stopping supported.

Things Didn’t Work

There are some other features and analysis I have tried but didn’t really improve the performance.

Customer Segmentation

I tried a KNN algorithm to split users into 4 different clusters based on the user level features I described above. The hypothesis is that users will vary in the order frequency, reordering rate, interests for different products etc. KNN helps me better understand users’ behaviors, but unfortunately, my performance didn’t improve much when I added the users’ cluster features into the model. The reason might be that the features that I use for clustering are already in the model. You can refer to my notebook “905_kmeans_cluster. ipynb” for details.

Association Rules

Association rules is a data mining technique. It analyzes user transaction data to discover which groups of products tend to be purchased together. I have found some strong association rules between certain products with their lift greater than 3 and confidence larger than 40%. For example, some yogurt and sparkling water in different flavors, such as Raspberry Yogurt and Blueberry Yogurt are often bought together. However, these rules only apply to products bought in the same order. They are more of inner-order rules, it’s not applicable for predicting future orders from past orders.

Future Work

Better ways to Find 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. That's why I also used the results of word2vec algorithm to identify substitutes. Check out Product Embedding for more details. However, word2vec is an indirect way of identifying substitutes and therefore will not be accurate enough. A more direct way would be finding pairs of products where the probability of both occurring in the same order is much smaller than the probability of each occurring separately. Namely, $P(AB) << P(A)*P(B)$. Also, we might also need to consider each person might have different substitute rules.

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.