Market Basket Analysis

Process and Learnings

Tiffany Chen | GA Data Science | April 2021

instacart

Agenda

- Project Overview
- Study Design
- General Trends
- Model Findings
- Conclusions*

*not very conclusive

Project Overview

Problem Statement

The goal of this project is to conduct a market basket analysis on a sample of Instacart users to better understand better customer purchasing patterns. Market basket analysis can help to analyze a wide variety use cases including predicting the likelihood that a user will buy again, try a product for the first time, or add a particular product to their cart next during a session. For this project, I will be focusing primarily on predicting which previously purchased products will be in a user’s next Instacart order.

Limitations

We are currently unable to characterize Instacart users as all personal user data are anonymized, and the data includes orders across many different retailers. The most we can uncover using the given dataset is segmented by time/date of purchase. The dataset provided represents a subset of Instacart's production data, which may be heavily biased. One example of this bias is that orders per customer are limited to 4-100 orders per customer.

Hypothesis

The hypothesis is that previous buying behavior predicts future buying behavior. Product purchasing habits are mainly a product of habit, indicated by patterns of re-purchasing everyday groceries, or a set cadence of order-placing.

Data Parameters

- The data consists of large sample of order histories, where each user has between 4 and 100 orders in the dataset.
- For each unique order, we know the following:
  - Products in the order + cart placement sequence
  - Time of day/day of week of order
  - Whether product is a reorder
- Modeling was performed on **top 50 percentile** of order volume **(>16 orders)** to optimize performance on top users
- Products purchased for the first time on each user's latest order were removed given lack of history

Solution Approach

- Models were fit on dataset of unique user-product pairs to predict likelihood of repurchase. When aggregated by user, the prediction represents an entire cart
- Each user-product pair classified as:
  - **Positive class:** the user purchased the product in their latest order
  - **Negative class**: the user had previously ordered the product but did not purchase in their latest order
- Data split into train, test, validation

Metrics Used

F1	ROC Curve	Briers Loss
Harmonic mean of precision and recall Used for imbalanced class distribution (0:0.9, 1:0.1), wanted to penalize FP and FN more than I wanted to reward TN	Plots TPR against FPR Diagnostic of classifier’s predictive ability, which I used primarily to compare performance across different models	Loss function to measure accuracy of probabilistic predictions, more appropriate for binary classifiers Doesn’t fare great with imbalance classes but does favor positive class

Features Data Dictionary

- **user_id**: *unique user*
- **order_id**: *sequential orders by user*
- **product_id**: *unique product per user*
- **reordered**: *Boolean, 1 if product was reordered by user*

Product Stats

`product_ordered_vol` : *Total amount of this product ordered*
`product_reordered_vol` : *Number of times product reordered across all users*
`product_ordered_once_vol` : *Number of times product ordered once across all users*
`product_ordered_twice_vol` : *Number of times product ordered twice across all users*
`product_order_twice_ratio` : *product_ordered_twice_vol/product_ordered_once_vol*
`product_avg_reorders` : *product_reordered_vol / product_ordered_once_vol)+1*
`product_overall_reorder_prob` : *product_reordered_vol / product_ordered_vol*

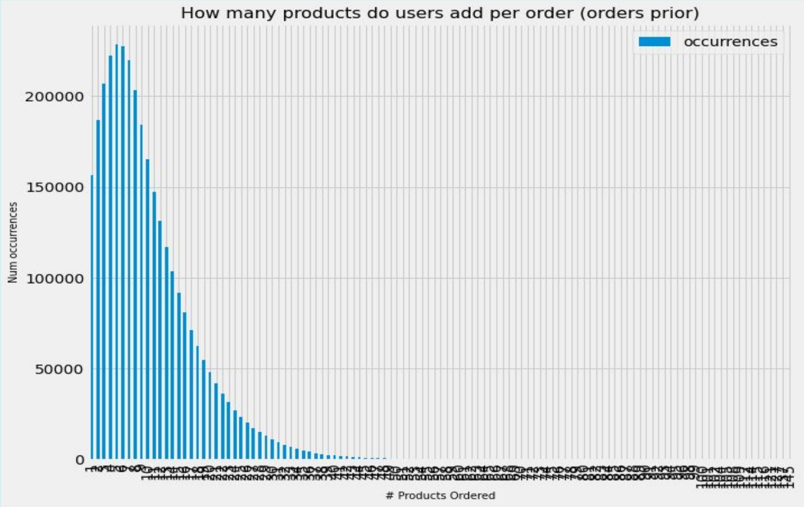
User Stats

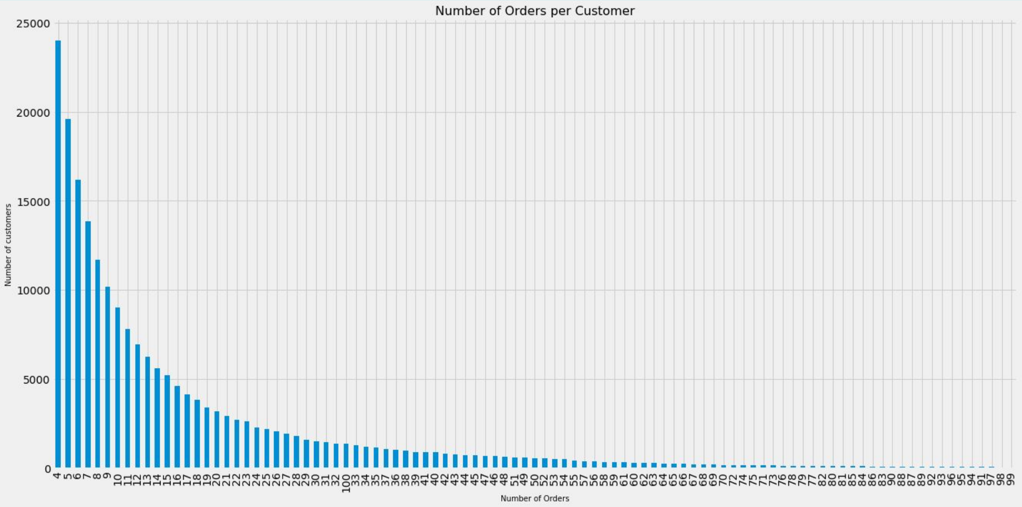
`user_total_products` : *Total number products ordered by user*
`user_total_products_reordered` : *Total number products reordered by user*
`user_unique_product_count` : *Number of unique products by user*
`user_unique_reorder_count` : *Number of unique reordered products by user*
`user_unique_product_perc` : *user_unique_product_count/user_total_products*
`user_unique_reorder_perc` : *user_unique_reorder_count/user_total_products_reordered*
`user_total_items_after_first_order` : *Number of products user ordered after first order*
`user_reorder_ratio` : *reorder_count/total_items_after_first_order*
`user_total_orders` : *Number of total orders by user*
`user_lifetime_days` : *Number of days each user has been active on Instacart*
`user_avg_days_between_orders` : *Average days between instacart orders*
`user_max_time_between_orders` : *Maximum days between instacart orders*
`user_min_time_between_orders` : *Minimum days between instacart orders*
`user_avg_cart_size` : *Average cart size by user*

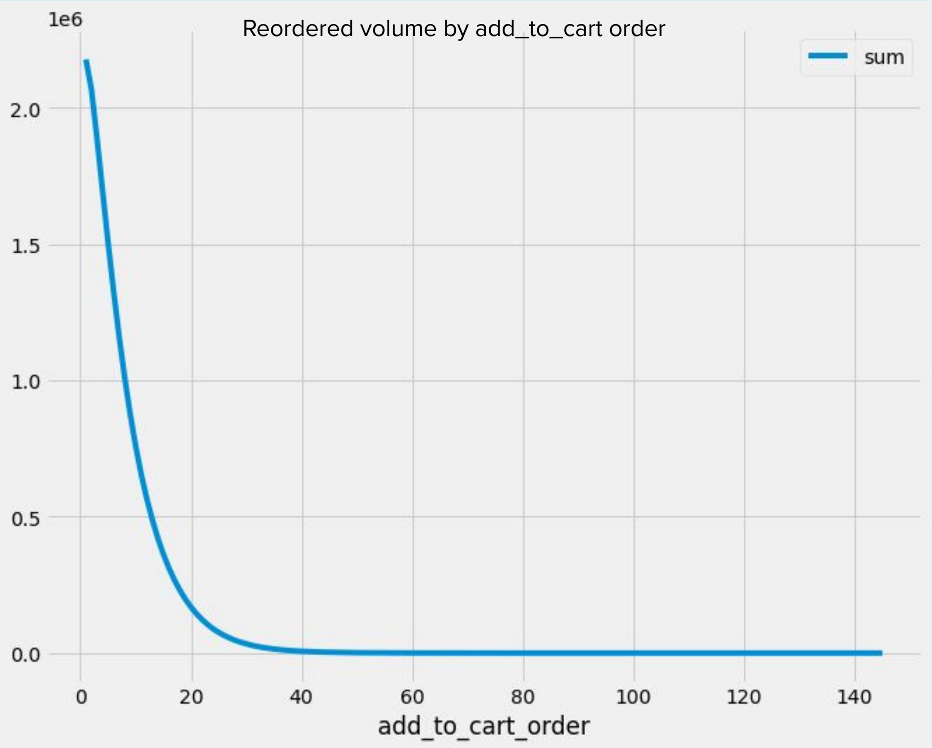
User-product Stats

`user_product_count` : *Total times product was ordered by user*
`user_product_first_order` : *First order placed containing product by user*
`user_product_last_order` : *Last order placed containing product by user*
`user_product_avg_basket_placement` : *Average 'add_to_cart_order' for product by user*
`user_product_order_rate` : *user_product_count/user_total_orders*
`user_product_reorder_rate` : *user_product_count / (user_total_orders-user_product_first_order +1))*
`user_product_last_time_product_ordered` : *number of orders since user last ordered a product*

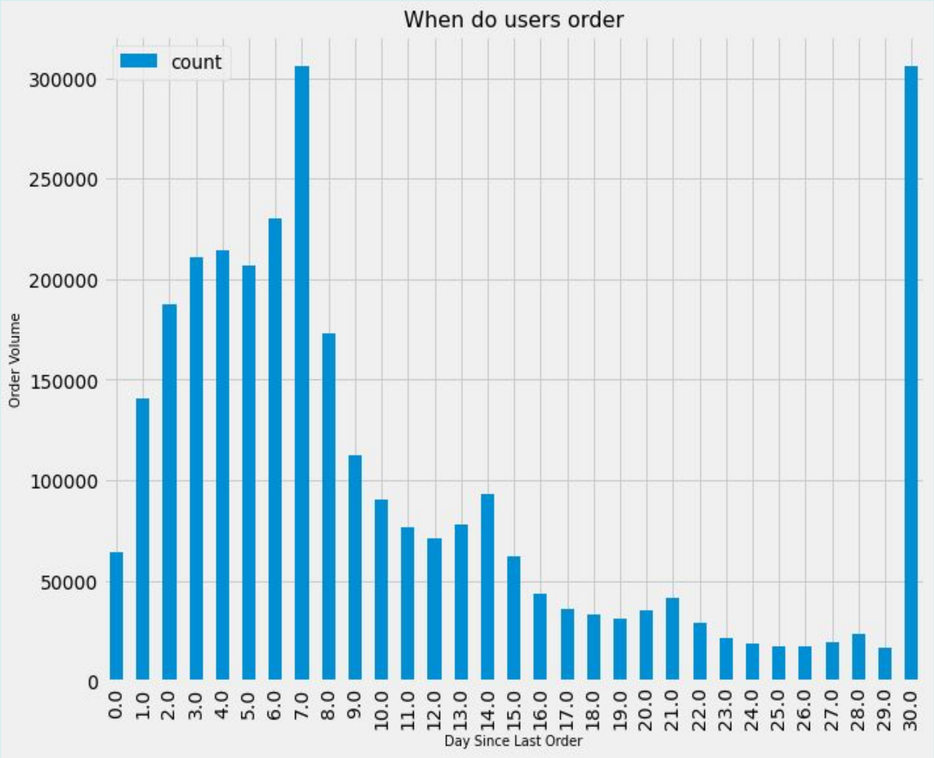
Some EDA Outtakes







1e6
Reordered volume by add_to_cart order
sum
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
120
140
add_to_cart_order



When do users order
count
Order Volume
300000
250000
200000
150000
100000
50000
0
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
24.0
25.0
26.0
27.0
28.0
29.0
30.0
Day Since Last Order

Modeling

Logistic Regression

Set-up

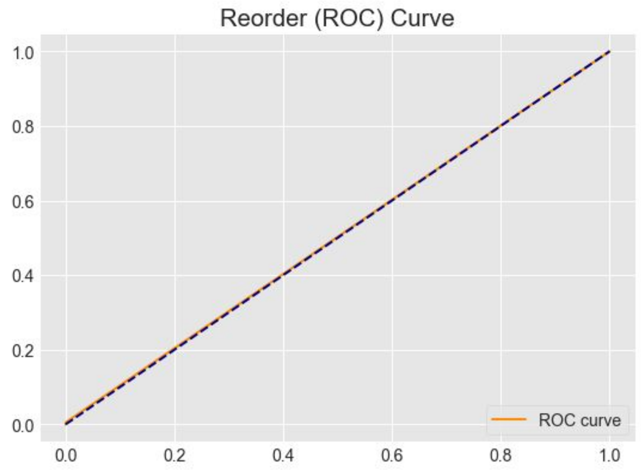
- Multicollinearity Check (drop columns with |correlation| > 0.8)
- Filter for users making above mean # of orders (16)
- Split T/T/V sets randomly by row
- T/T/S on Train set

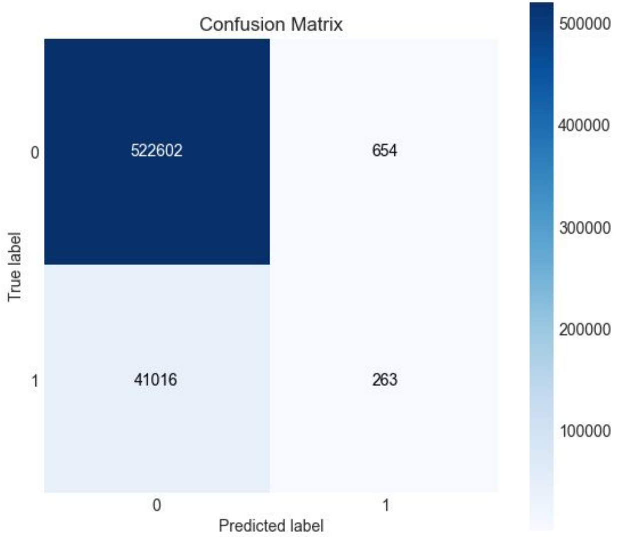
Tuning (all things I tried)

- Scaling features minus ID columns using **Standard Scalar** and **Column Transformer**
- Dimensionality Reduction with **PCA**
- **Grid Search** with **cross validation** to find optimal class weights
- Grid Search with CV for C regularization tuning
- Probability Threshold Tuning
- Feature Selection investigation using **SelectKBest,** recursive feature elimination **(RFE)**, **SelectFromModel**

Test

- Test final model on validation, test, and whole

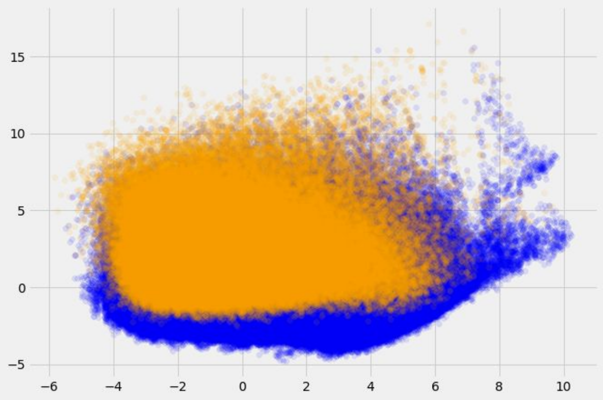




Further

What was ultimately rejected in tuning

- PCA- use of principal components didn't establish a clear enough boundary between classes for this to be very useful



- Feature Scaling - didn't apply scaling because didn't end up using regularization term, and ultimately didn't impact modeling performance

- On Feature Selection
    - Not used for model optimization
- On interrogating the similarity between train, test, validation performance -
    - Redid splitting of train/test/validation datasets by user_id grouping instead of row- originally was concerned that users split up into different sets would bleed information to these 'unknown data sets'
    - Ultimately had little impact

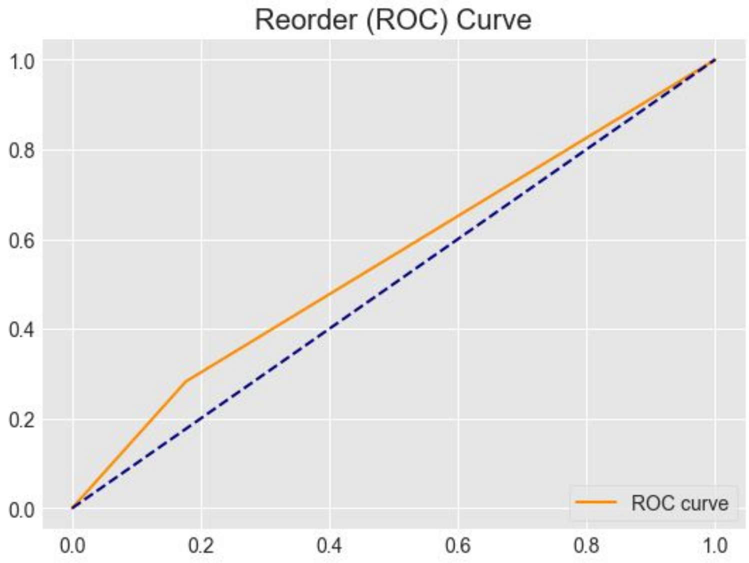
Conclusions

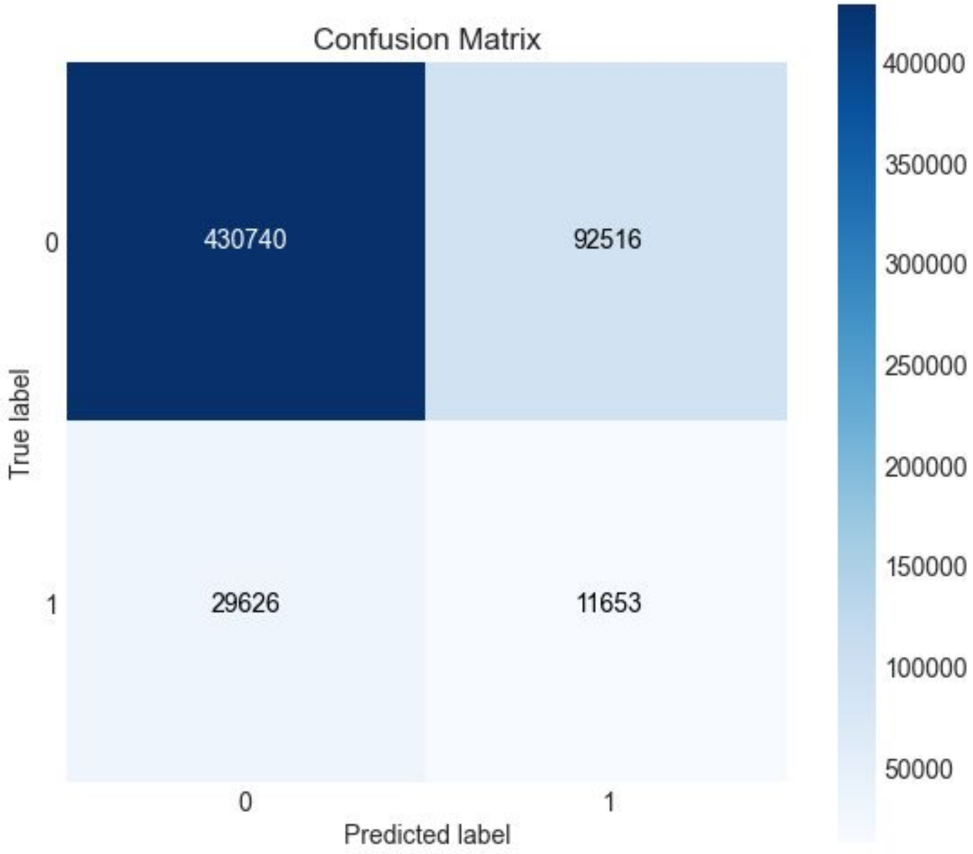
- For the purpose of answering this question, the logistic regression model fared poorly - had much higher success predicting based on user-product reorder rate
- Feature experimentation led to discovery that model was highly successful in solving other questions

Results

Final Train AUC: 0.5527
Final Train F1: 0.1602

Overall Test AUC: 0.5518
Overall Test F1: 0.1599

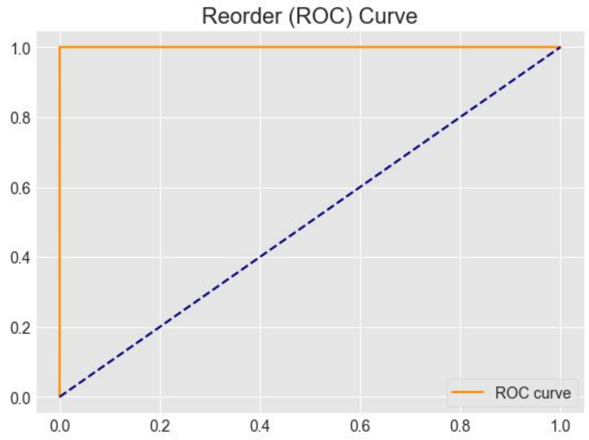




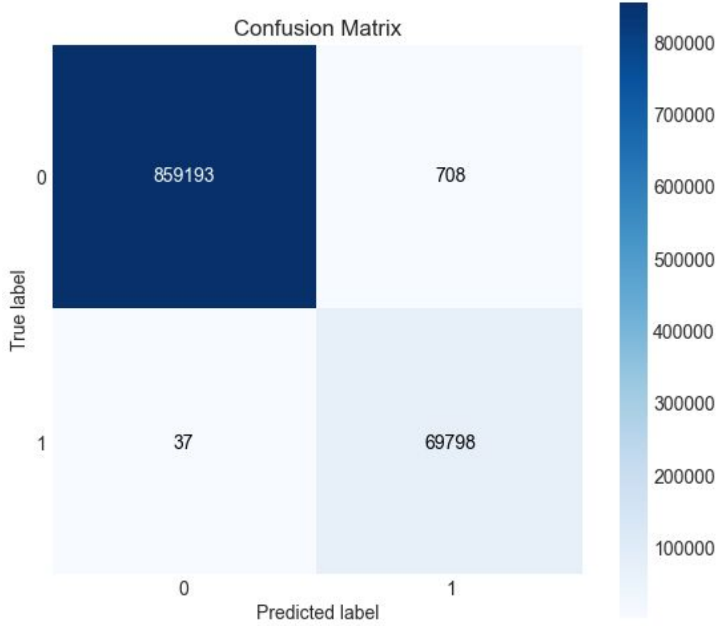
Tree based Models

Fitted **Random Forest** classifier and **XGBoost** classifier and did feature discovery using **plot_importance**
Very little hyperparameter tuning needed, as both tree based models performed incredibly well on train, test, validation, and whole data (below metrics are oos data)

Random Forest	XGBoost
F1: 0.996	**F1:** 0.996
AUC: 0.999	**AUC:** 0.999
Loss: 0.00057	**Loss:** 0.00057



- XGBoost tuned for **max_depth, n_estimators, and learning_rate** using **RandomizedSearch** and **GridSearch** to try to mitigate overfitting if it was happening, but results were largely the same

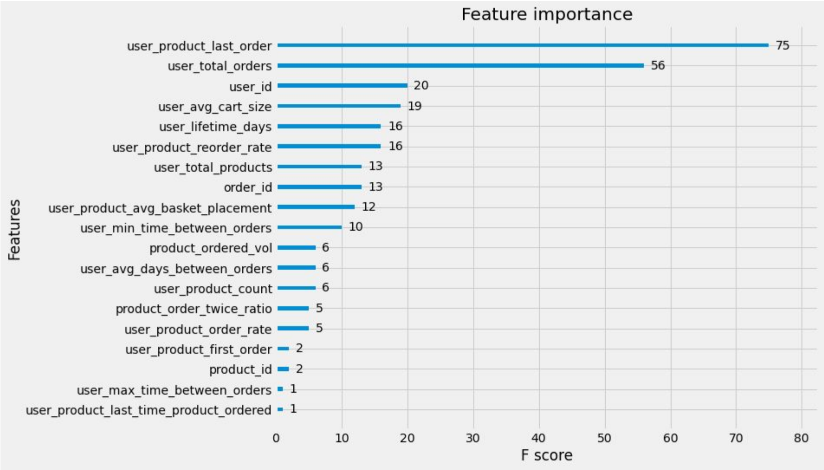


Because im super bad at all this

Practicing Skepticism

- Interrogated the true efficacy of the models by first revisiting engineered features to evaluate:
    1. Intuitive relationships
    2. Missing features
    3. Feature bleeding

1. Examining logical feature relationships to target- Initial run of model placed a lot of importance on *user_unique_product_perc* - % unique products ever ordered by user, which intuitively doesn't seem like it should be that impactful of a metric
2. Variables unavailable at time of data collection- Missing a lot of context due to nature of dataset - timing, seasonality, characterizing variables
3. Any calculations that could have bled information about the target during feature engineering- Re-splitting T/T/V data
   Otherwise ongoing



From final XGBoost run

Does model just respond well to big + loud datasets?

- Issue of high cardinality in a lot of features, which trees tend to favor

Establish a **Negative Control** by introducing noise

1. Randomized target against sample of training data using **random** and **randint**
2. For good measure, changing up some feature values (chose important ones) using **random.choice**

Good news is that model did not find any meaningful relationships in the noise, with an avg AUC of .5 over multiple runs

Conclusions

Features are king!!

This modeling process turned out to be more of an exercise of feature engineering and feature exploration. I learned a lot about how each feature or combinations of features could impact the model. It turns out, the most useful features that my models depended on mapped to particular **user behaviors**, whether it be patterns in purchasing behavior or relationships with the products themselves.

Interest areas:

- Relationships with product type (perishables vs canned) or characterizing patterns for infrequently ordered items
- Better characterizing product displacement and “switching” behaviors

Other opportunities

Since this model was fitted on users making above the mean number of instacart orders in their lifetime, it would be interesting to see how the model degrades for users making <16 orders. Of the features identified as important across all the models, most of them tethered around ‘reorder’ statistics (*user_product_last_time_product_ordered*, *user_product_reorder_rate*, *user_product_last_order*), which would be less available in this segment + require some ‘None’ handling

While the logistic regression largely underwhelmed in its performance for the given problem, surprisingly it was very capable in solving a different problem. In my tuning, when user_id is taken out as a feature, the model performs much better, in that it is able to predict products that are reordered with each successive order (ie: order #5 across all users will reorder XYZ items), since there is some interpretability in the sequential nature of the order_id variable compared to the user or product_id variables. This wasn’t heavily tested so it would be interesting to test how far this performance actually extends.

🌮🌮🌮

thank you

Stay in touch!

linkedin.com/in/tiffanychen44

IG: @ti_panini