Springboard Data Science Career Track Capstone One – Buy Again Market Basket Analysis

Rose Zdybel

Introduction

Suppose you use an on-line shopping service for frequently purchased items such as groceries. A helpful feature is for the user interface to assist you in generating the order. For example, by presenting the list of items you commonly order, such as milk, eggs, or certain produce so you don't have to search for them. It would be even nicer if the recommendations are not cluttered with items you infrequently order and presents them when you are likely to repurchase them, or at least prioritizes them with the most likely items at the top of the list.

Many retailers provide this type of functionality in on-line websites. However, they are always looking for ways to improve the predictions. That is the case in Instacart's Market Basket Analysis Kaggle competition.

This project implements a baseline approach for predicting items a customer is likely to reorder in an on-line grocery shopping session. Information about a user's past shopping sessions is known, as is information about other customer's shopping sessions.

Why does the retailer care? Knowledge about product demand and customer purchasing habits allows them to determine which products and quantity to stock, keep customers coming back by improving their experience, and potentially increase sales.

Overview

We use data from the Kaggle Instacart Market Basket Analysis competition .

This report describes a solution to the buy again prediction issue. It explores the use of some alternative models but focuses on using the Light Gradient Boosting machine learning algorithm to make the predictions.

Since the data is from a Kaggle competition, the data is clean and healthy. The process involves verifying the data health, exploring data characteristics, formulating the problem, generating model features, choosing then tuning the model, and determining predictive capability.

Data acquisition and verification

In this section we describe the data source and the verification steps.

DataSet

Data used for the analysis is from the Kaggle Market Basket Analysis challenge, sponsored by Instacart.

Kaggle Link

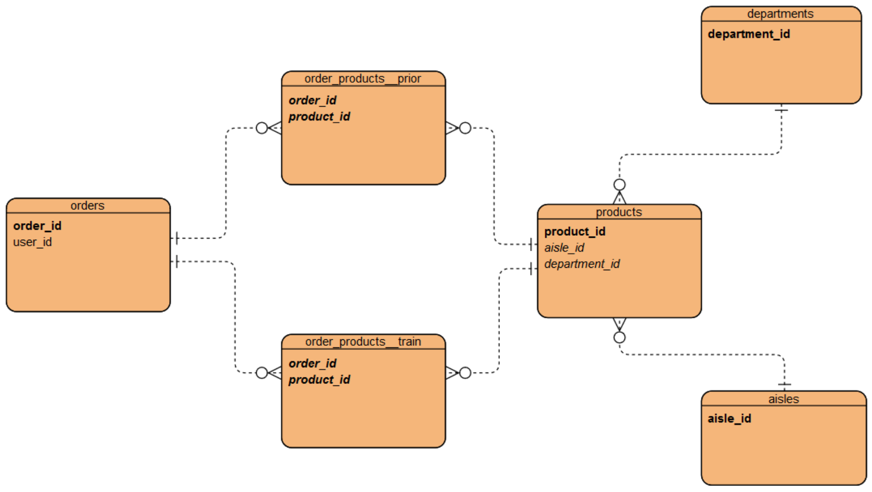
https://www.kaggle.com/c/instacart-market-basket-analysis/overview

Sample Data

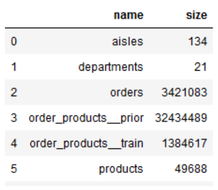
For the Kaggle competition, Instacart provides the data in a normalized form in the data structure sense; foreign key values for data in one file are used to reference data in other files.

The core data is in the orders and order_products files. Other files contain lookup information for product details, aisle, and department. The data linkage is illustrated in the diagram below.

Figure 1 - Source data linkage



The size of the data is summarized in the table below. There are 3.4 million orders for approximately 206,000 customers. The order-products are contained in two datasets, for modeling purposes: the order_products__train file contains the results we are trying to predict during development; the order_products__prior file contains information on user's prior orders, for both the users in the training and competition submission data. **NOTE**: for the competition submission, the order characteristics for the submission orders are in the orders data table; however, we of course do not know the order_products information for those orders, since that is what we are trying to predict.

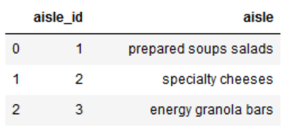


	name	size
0	aisles	134
1	departments	21
2	orders	3421083
3	order_products__prior	32434489
4	order_products__train	1384617
5	products	49688

Figure 2 Number of Records in source files

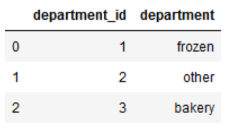
Sample data from each of the source files is shown in the figures below.

Figure 3- aisles



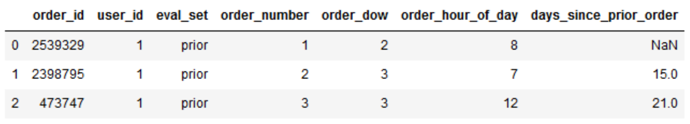
	aisle_id	aisle
0	1	prepared soups salads
1	2	specialty cheeses
2	3	energy granola bars

Figure 4 - departments



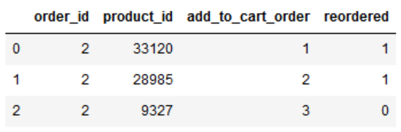
	department_id	department
0	1	frozen
1	2	other
2	3	bakery

Figure 5- orders



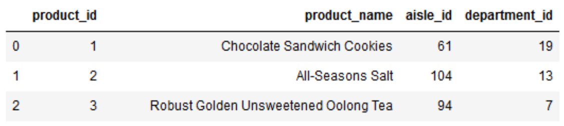
	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order
0	2539329	1	prior	1	2	8	NaN
1	2398795	1	prior	2	3	7	15.0
2	473747	1	prior	3	3	12	21.0

Figure 6- order_products (__train, __prior)



	order_id	product_id	add_to_cart_order	reordered
0	2	33120	1	1
1	2	28985	2	1
2	2	9327	3	0

Figure 7 - products



	product_id	product_name	aisle_id	department_id
0	1	Chocolate Sandwich Cookies	61	19
1	2	All-Seasons Salt	104	13
2	3	Robust Golden Unsweetened Oolong Tea	94	7

Data verification

We describe the steps taken to ensure the data is healthy. Since the data is from a Kaggle competition, we anticipate that the data would be in good shape as provided. Nonetheless, we do checks to validate the data integrity. We check integrity of linkage values used to associate records between source files. We also check for missing values, outliers, and basic value ranges.

Linkage values

After loading each file into its own dataframe, we check to ensure that there are no missing values that link the data. For example, we check that all **order_ids** in **order_products** are in the **orders** dataframe. We check integrity for all other linkages shown in the linkage figure above such as **aisle_id,** and **department_id** for **products**, and **product_id** for **order_products**.

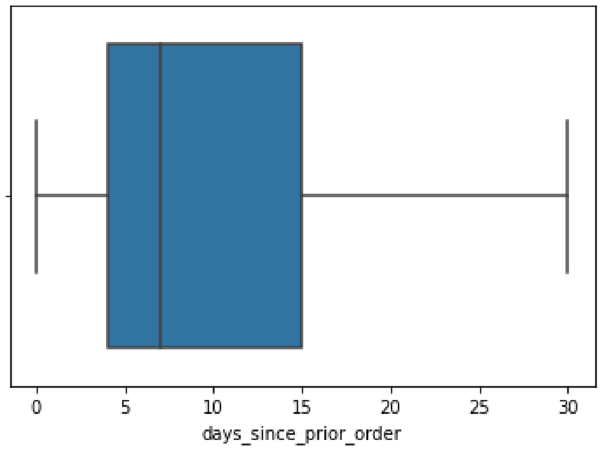
Missing values

We check each dataframe for missing values.

The only column that contains missing values is the orders.days_since_prior_order . This is to be expected, since the first order for a customer does not have a prior order.

Outliers

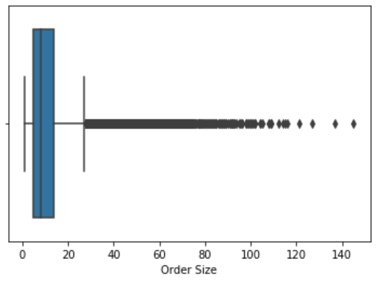
For outliers, we check the two numeric values that are not either categorical or sequential:

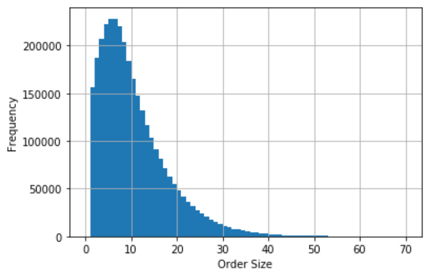
- *Orders.days_since_prior_order*
    - The boxplot below (and describe() method) indicates that there are no outliers, with a median value of 7 and a max value of 30.
    - 

- Order Size, based on the column:
    - *Order_product__prior.add_to_cart_order*

  The Order Size is not explicitly stored in the data. We derive it from the number of items in a cart. We use aggregation to get the max value of the add_to_cart_order column for each order, and then plot a boxplot and histogram of the order size. The distribution is skewed right, but the values are not unexpected, with a maximum order size of 145 items.

  The median is 8 items and the 75th percentile is 14 items.





Other

We do other basic checks on values in fields such as of DayOfWeek and HourOfDay.

Data exploration and inferential statistics

The goal of the data exploration step is to understand the data characteristics and generate hypothesis as to factors that may have predictive value.

In the interest of not inflating the length of this report, we do not go into each characteristic in detail. Instead, we give examples of the exploration, list the characteristics explored, and in the modeling section provide a list of the features used in the model. The code for the entire data exploration process can be found in the EDA Jupyter notebook.

We do data exploration at two levels:

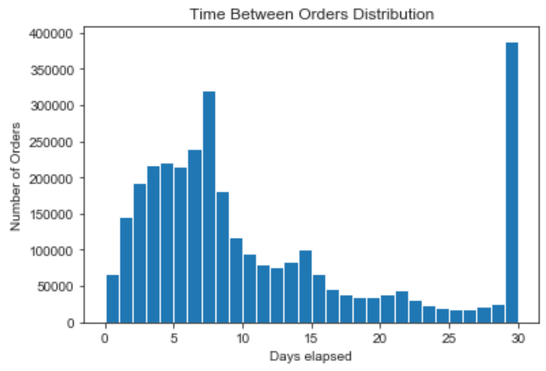
1. Baseline exploration
    a. Explore basic characteristics of the data.
2. Model-focused exploration
    a. Explore data characteristics that may influence product reordering, the outcome we are interested in predicting

Baseline exploration

Baseline exploration involves looking at distributions and relationships of the raw data. In this section we discuss two representative characteristics: the distribution of Time Between Orders; and distributions for Order Size by days since prior order. We provide a list of other characteristics for which additional details of the entire analysis can be found in the associated EDA Jupyter notebook.

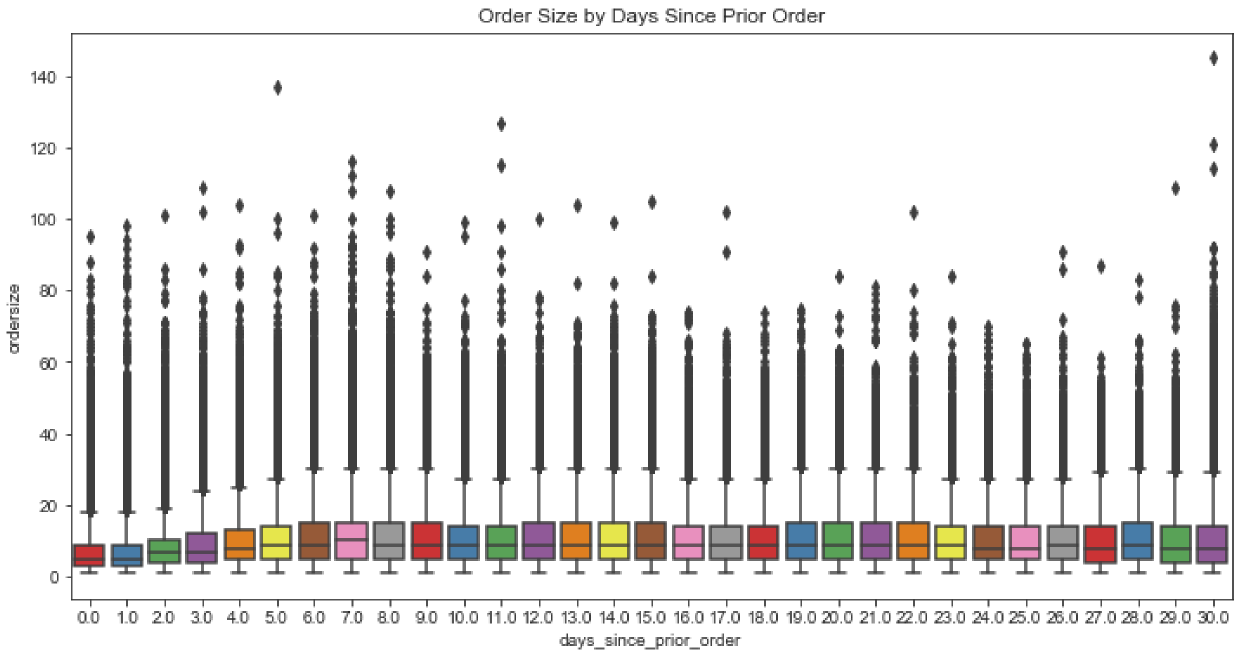
Time Between Orders

The bar chart below shows that the most frequent time interval between user's orders is 30 days; however, it appears that Instacart truncated to 30 days intervals greater than 30. Other than that, that most frequent time interval between user orders is 7 days.



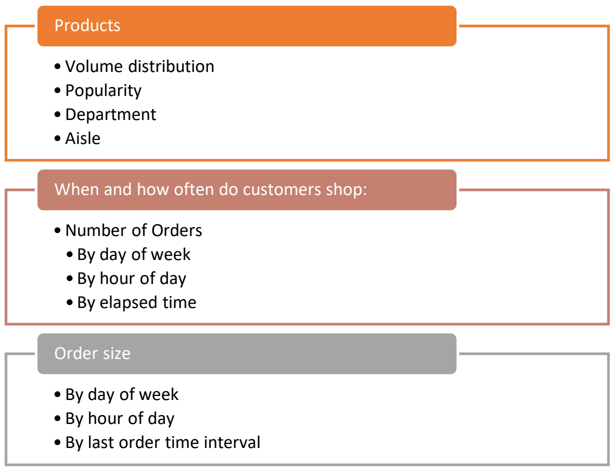
Order Size by Days Since prior order

The figure below shows a box and whisker plot of the order size for each value of days_since_prior_order. The general trend is that order size increases as the time interval increases from 0 to 7 days, then stays relatively constant at a slightly decreased size from 8 to 30 days.



List of other baseline characteristics

Other baseline characteristics explored in the Jupyter notebook are listed below.



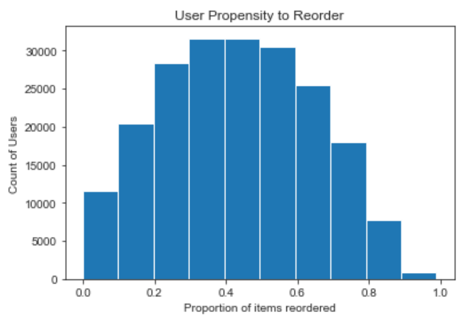
Model-related EDA

Model-related EDA focuses on characteristics that may have predictive value. We focus on reordering trends and explore them in relationship to things such as individual customers and hour of day and day of week purchases.

In this section we discuss two representative characteristics: customer propensity to reorder; and reorder proportion distributions by hour of day.   We provide a list of characteristics for which additional details can be found in the associated PowerPoint Presentation and in the associated Jupyter notebook.

Customer Propensity to Reorder

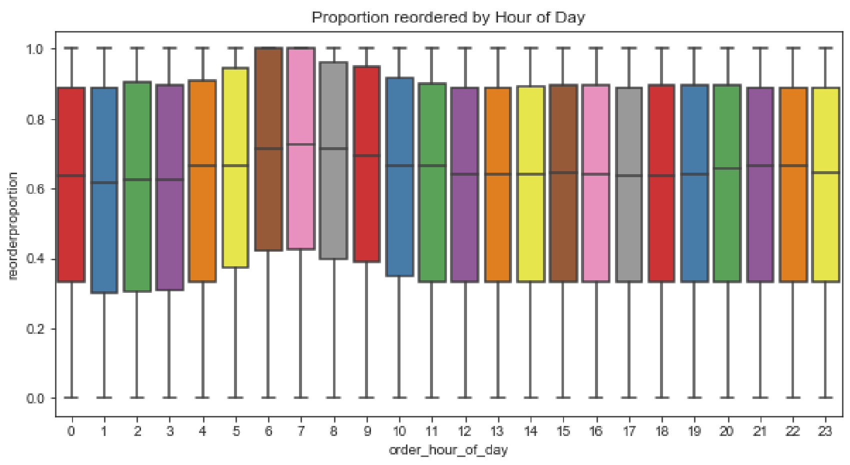
The chart below shows a distribution of user counts and the proportion of their purchased items that were reordered products. The reorder proportion ranges from less than 10% to over 90%, with the majority falling in the 30% to 60% range.



Reorder Proportion by Hour of Day

The figure below shows, for each hour of the day, a box and whisker plot of the proportion of items that are reordered for orders placed in that hour of the day.

The proportion of basket items reordered varies by hour of day, with orders early in the morning (between 5 and 9 o'clock) more likely to have a higher proportion of reordered items



List of Other model-related characteristics

Other model-related characteristics explored in the Jupyter notebook are listed below.

Customer

- Propensity to Reorder

Frequency and time effects on Reordering

- Reorder Proportion
  - By day of week
  - By hour of day
  - By elapsed time

Item reorder frequency

- by Dept

Most Reordered

- By Product
- By Department
- By aisle

Inferential statistics

We illustrate ways in which we could apply inferential statistics to data (a course requirement).

We apply inferential statistics to determine if there are statistical differences between data characteristics. The entire analysis can be found in the associated Inferential Stats Jupyter notebook. We summarize the process here, without presenting the details.

- For model training:

    - When splitting data for model development into training and test sets, we check selected statistics to ensure that they are not statistically different between the two subsets.
- For model predictive capability:
    - Illustrate the use of statistics to determine if categorical features influence outcome:
        - Check to determine if differences in reordering behavior are statistically significant for orders placed during different hours of the day. We use selected hours to show that they can be significant; for example, differences for hours 7 and 14 are significant.

Modeling

This section describes the model development process which consists of formulating the problem into a usable form, determining which algorithm(s) to consider, developing the model, and evaluating the results.

The subsections in this section contain discussions of these topics:

- Kaggle vs standard machine learning terminology used for splitting test/train data.
- Restructuring the data to formulate the problem, since the data is not provided in a directly-usable format.
- Metrics used for model evaluation
- Algorithm choice
- Hyperparameter tuning using 5-fold cross validation.
- Model training and prediction capabilities.

Modeling Preparation

In this section we discuss the preparation steps for model-selection and development. The subsections consist of test/train data terminology; model formulation/data restructuring; and evaluation metric.

Test/train/submission data terminology

We clarify how we use test/train data terminology in this report, since the Kaggle competition usage conflicts with standard machine learning terminology.

In the competition terminology, 'test' data is what is to be submitted to the competition, and for which we do not know the outcomes. In contrast, typical machine learning uses 'test' to refer to data for which we know the outcomes and use it to validate the trained model during model-development. To avoid confusion, in this report we refer to the competition 'test' data as 'submission' data. We refer to their 'train' data as the 'development' data. Within the 'development' data, we use machine learning standard terminology of train/test for model-development purposes; for example, we split the 'development' data into test/train sets.

Model Formulation / Data Structuring

The goal is to predict which items a user previously purchased are in that user's last order.

The data, as provided in the order_products__train dataset, gives the items in the last order. Since it does not explicitly denote all previously purchased items, we generate that from prior purchases (essentially a cross product of each user and the items that user purchased in prior orders), and use the value '1' to indicate if the item was purchased in the final order, and a zero to indicate not purchased. This is then used as the basis for a binary classification problem.

We augment the data with features that we hypothesize could predict product reordering behavior. Many of the features are derived as summary statistics from prior purchases and are described in the 'Feature generation' section below.

Evaluation metric

The Kaggle competition uses the F1 score as the ultimate metric for model performance. However, we do not directly use it for model selection since the data is imbalanced (in the final formulation, there are more unpurchased items than purchased) and binary classification problems are biased towards the dominant outcome.

So, for model selection and parameter tuning, we use the ROC-AUC score. To counteract data imbalance, we need to choose the threshold used for translating the model output probabilities to binary predictions [https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/]. We choose the threshold that gives the best F1 score.

Feature generation

For model features, in addition to directly using some of the data provided in the source data, we 'engineer' other statistical features from aggregated data and based on the exploratory data analysis discussed previously in the Data Exploration section. We provide further details in this section.

In general, we group the features by their association with data entities (users, orders, products, departments, aisles). A feature can be associated with a single entity, or multiple. For example propensity to reorder (u_reorderMean) is a user feature while user-product features include proportion of orders a product is purchased by a user (up_order_proportion) and average product position in cart for a user (up_avg_pos_in_cart).

Features used in the final model are summarized in the table below

Group Feature Name	**Description**
Aisle	
a_reorder_proportion	proportion of purchases from aisle that are reordered items [count(aisle product reordered) / count(aisle product purchased)]
aisle_id *(categorical)*	aisle identification
a_count	total number of times a product was purchased from aisle by any user
Department	
department_id *(categorical)*	department identification
Order	
order_dow	order day of week
order_hour_of_day	order hour of day
Product	
p_count	total number of times product was purchased by any user
p_reorder_proportion	proportion of purchases of product that are reorders for all users [count(product reordered) / count(product purchased)]
product_id	product identification
days_since_prior_order	days since prior order for user
User	
order_number	order sequence for user
u_reorderMean	overall proportion of items purchased that are reordered items by user [count(user reordered items) / count(user purchased items)]
user_id	user identification
User-Order	
uo_avg_day_of_week	avg day of week user places orders
uo_avg_days_between_orders	avg days between order for user
uo_avg_hour_of_day	avg hour of day user places orders
uo_totalorders	total number of orders for user
User-Product	
runningdays (up_days_since_last_purchase)	number of days since user last purchased the product
up_avg_pos_in_cart	avg position of product in cart for user
up_order_proportion	proportion of orders user purchases product [count(orders user purchased product) / count(user orders)]

Model-selection

From on-line discussions, we have information indicating gradient boosting models work well for this problem. However, for sake of completeness, we briefly look at other candidate classification models. We compare them using the model default parameter values and ROC-AUC.

We discount Logistic Regression due to technical difficulties in applying to this data, such as its restriction of non-correlation of features. In our case the generated features are likely to be correlated, and decision-tree based models are better suited for this. So instead, we compare random forest and gradient boost algorithms.

Specifically, the models we consider are: Random Forest, Scikit-learn GradientBoostingClassifier (GB), and LightGradientBoost (LightGBM).

Model comparison results

We compare the models using the ROC-AUC score with the default hyper-parameters for each model. We also consider train time. The following two figures show the ROC-AUC graphs and the ROC-AUC summary statistics, respectively.

Even though the two boosting models gave similar scores, the Light variation has a strong advantage in that it took only 55 seconds to train vs approximately 30 minutes for the GradientBoostingClassifier (and 2 minutes for the RandomForest). Therefore, we proceed with fine tuning and development of the model using LightGBM.

Figure 8- ROC AUC Model comparison

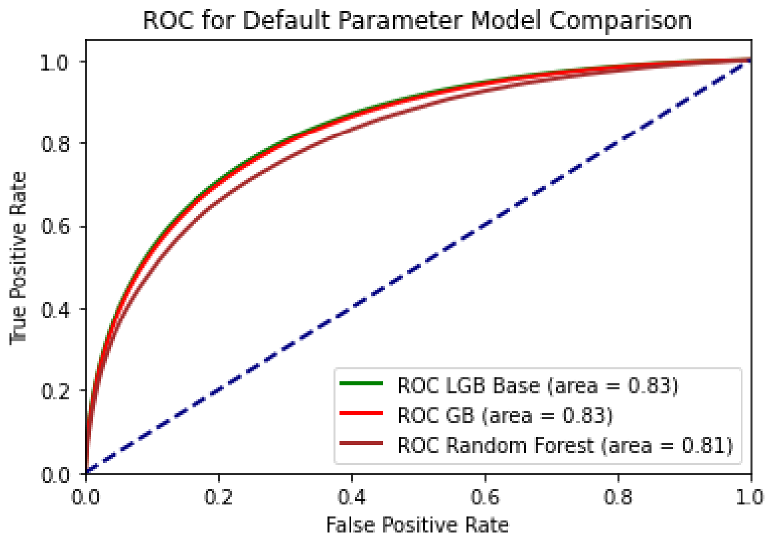


Figure 9 - Model comparison summary statistics

Model	ROC-AUC	Train time (mm:ss)
Random Forest	0.81	02:00
GradientBoostClassifier (GB)	0.83	30:00
Light Gradient Boost (LGB)	0.83	00:55

Model Development – LightGBM

The process used to develop the model involves the following steps:

1. **Split data**. We split the data into train and test portions. Since the dataset is fairly large, we use an 80/20 split for train/test proportions. We split by order_id, and then the data associated with the orders' users.
2. **Train model.** On the training data (of course).
    a. **Tune hyperparameters.** The LightGBM has many parameters. We follow the process outlined in this link to tune the hyperparameters, using the following options:
        i. **RandomGridSearch.** Since there are many combinations of parameters, we use a random search grid, rather than a comprehensive grid search. In a future project, we will consider using the Bayesian method described in the link, but for this project most parameter choices did not have a noticeable effect on the ROC-AUC score, so we did not pursue the Bayesian approach.

ii. **5-fold cross validation.** We use 5-fold cross validation (CV) for each set of parameters chosen from the grid, and the mean ROC-AUC score as the evaluation metric in choosing the best parameters.

b. **Train with best hyperparameters.** We now use the 'best' parameters identified in the previous step to train the model using the entire training data set (in the hyperparameter tuning step, it was sectioned into 'folds').

3. **Choose threshold.** As discussed in Evaluation metric section, the data is unbalanced, so we need to choose the discrimination threshold that maximizes the F1-score evaluation metric.
4. **Predict Test data.** The trained model is used to make predictions for the hold-out test data.

Results

We discuss the relevant results from the steps outlined above.

Hyperparameters/Learning Rate/num Trees

The input values used for the Random Grid search are shown below. We searched a limited number of combinations (10), since it was computationally intensive, and for the combinations it selected, there was little noticeable effect on the ROC-AUC score.

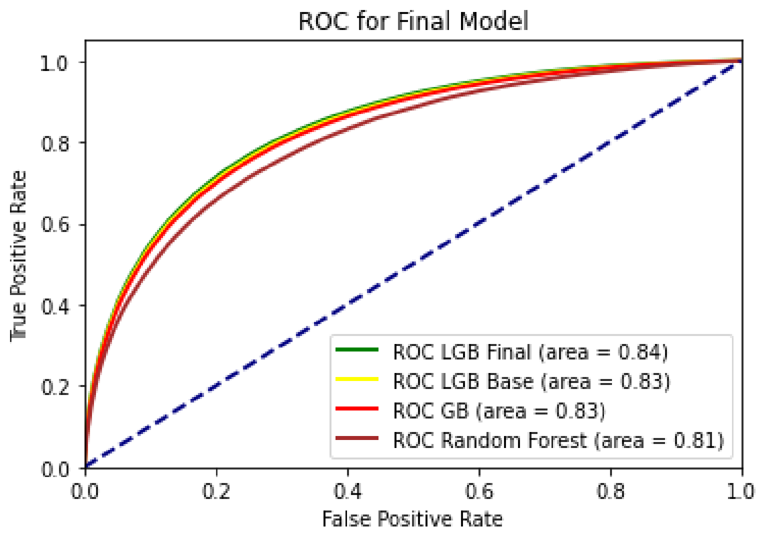
```
param_grid = {
    'boosting_type': ['gbdt'],
    'max_depth' : [7],
    'num_leaves': list(range(20, 150, 10)),
    'learning_rate': list(np.logspace(np.log10(0.05), np.log10(0.5), base = 10, num = 10)),
    'subsample_for_bin': list(range(20000, 300000, 20000)),
    'min_child_samples': list(range(20, 500, 50)),
    'colsample_bytree': list(np.linspace(0.6, 1, 10)),
    'subsample': list(np.linspace(0.5, 1, 10)),
    'n_estimators' : [25, 50, 75]
}
```

The limited search 'best' parameters are:

```
best_clf = lgb.LGBMClassifier(boosting_type='gbdt',
                    class_weight=None,
                    colsample_bytree=0.6,
                    importance_type='split',
                    learning_rate=0.2997421251594704,
                    max_depth=7,
                    min_child_samples=270,
                    min_child_weight=0.001,
                    min_split_gain=0.0,
                    n_estimators=75,
                    n_jobs=-1,
                    num_leaves=60,
                    objective=None,
                    random_state=None,
                    reg_alpha=0.0,
                    reg_lambda=0.0,
                    silent=True,
                    subsample=0.9444444444444444,
                    subsample_for_bin=280000,
                    subsample_freq=0)
```

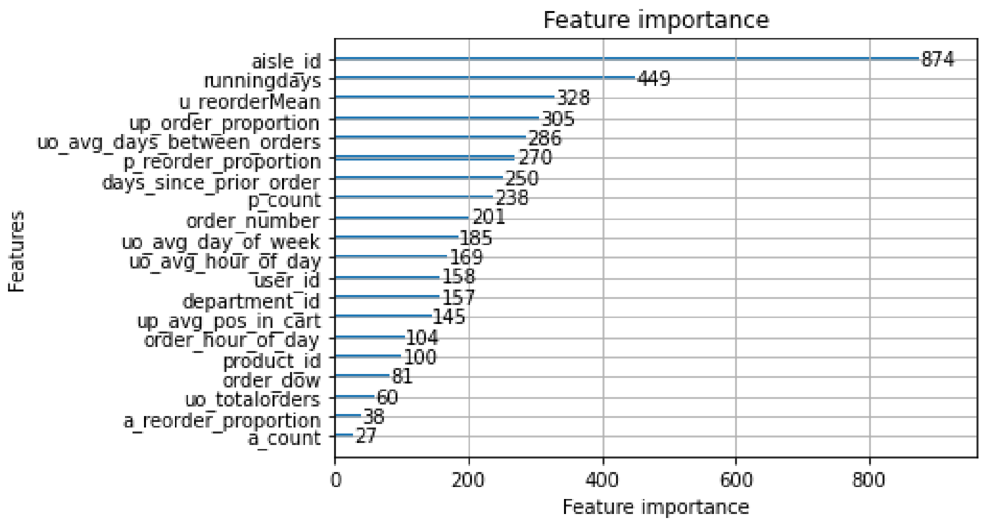
ROC Curve

The figure below shows the ROC curve, corresponding to the 'best' hyper parameter combination, along with the baselines we did earlier in model-selection. We see that the 'tuned' **LBG Final** model gives a slight improvement over the **LGB Base** default parameters model.



Trained Model – Feature importance

The feature importance grid shows the number of times a feature was used as the branching criteria in the trained model. Aisle_id, runningdays, u_reorderMean, and up_order_proportion are at the top of the list and seem reasonable. Those at the bottom of the list appear not to have much predictive value or are possibly redundant with others.



Trained Model – Example tree splits

In this section we show examples of the branching criteria the algorithm is choosing. The content is presented for informational purposes, for sanity checking, and validation purposes in case there would be unexpected results.

The first three figures below show snapshots at various levels of detail of the first decision tree generated by the model. We see that the splits are predominated by the features listed at the top of the Feature Importance graph (shown above): aisle_id, runningdays, u_reorderMean, and up_order_proportion.

The fourth figure shows the 23rd tree in the sequence. It shows that the splits are based on features further down in the importance list, such as uo_avg_days_between_orders, days_since_prior_order, and up_order_proportion.

Figure 10- Tree 1 View a

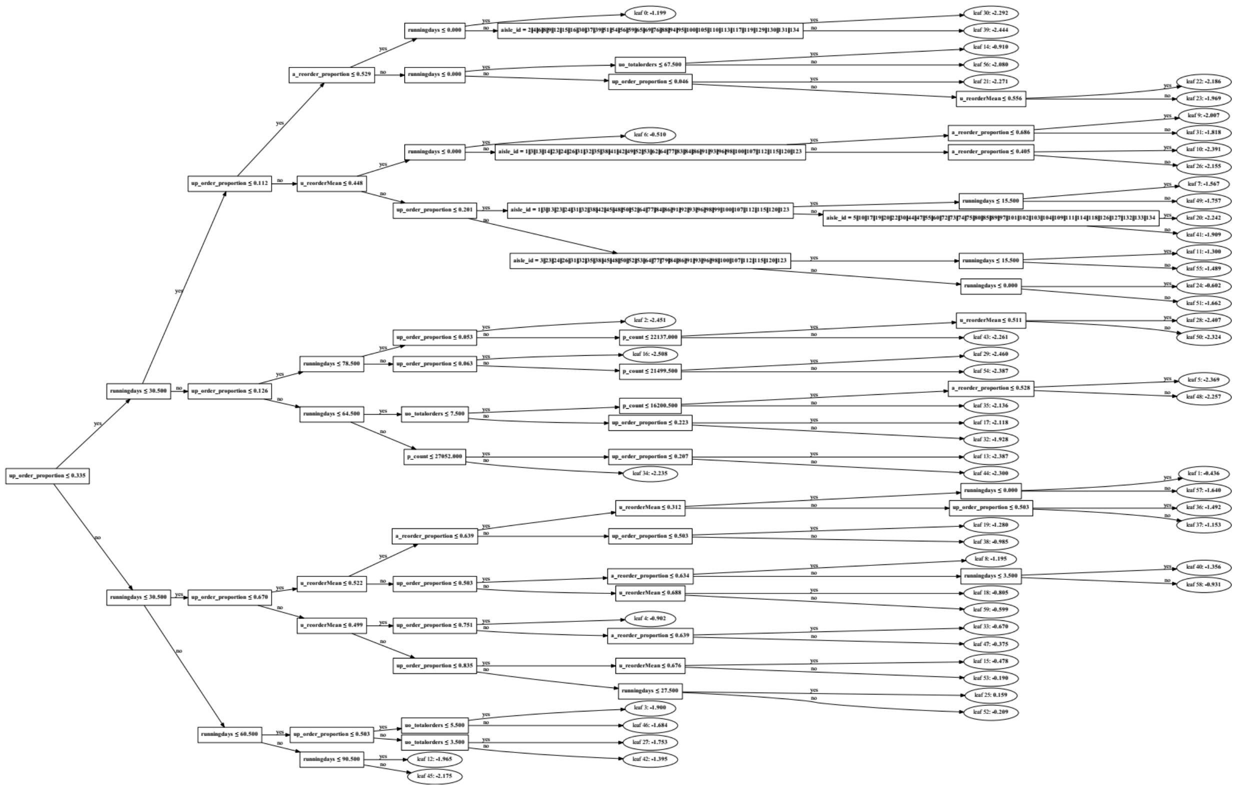


Figure 11 - Tree 1 View b

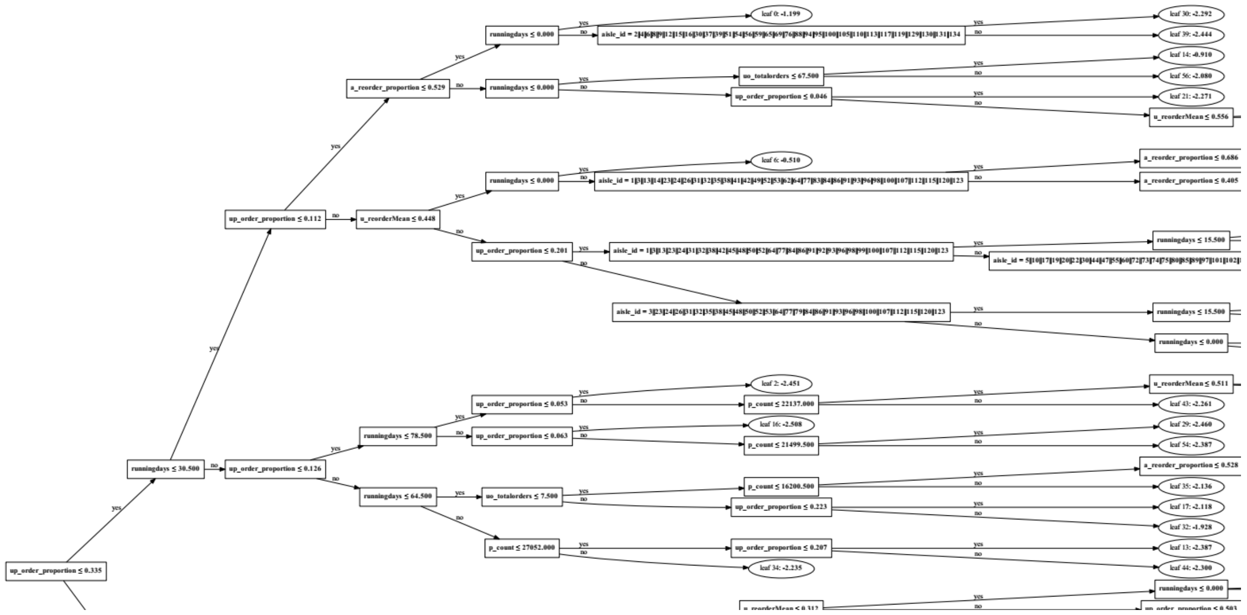


Figure 12 - Tree 1 View c

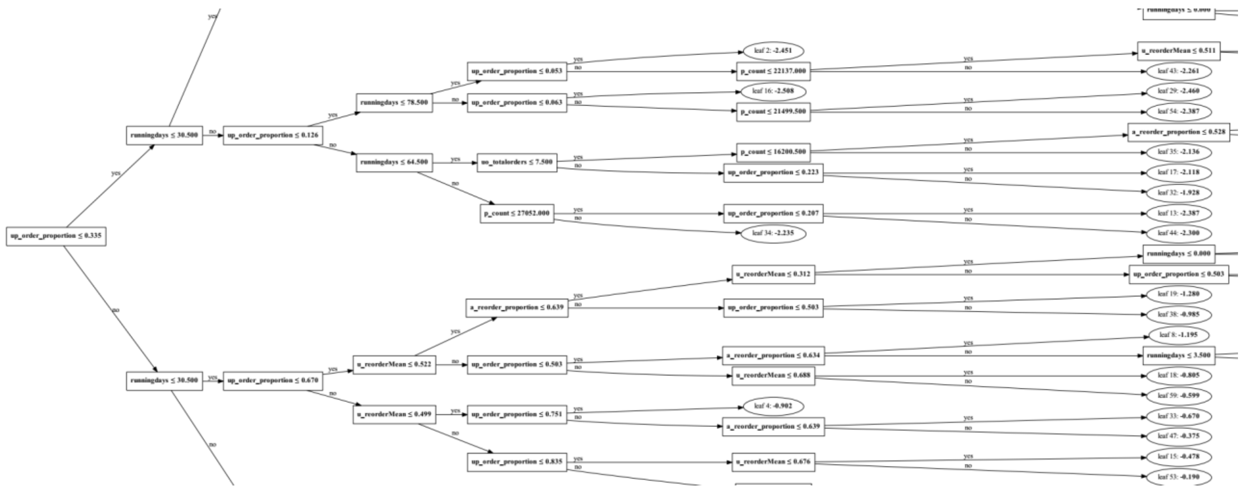
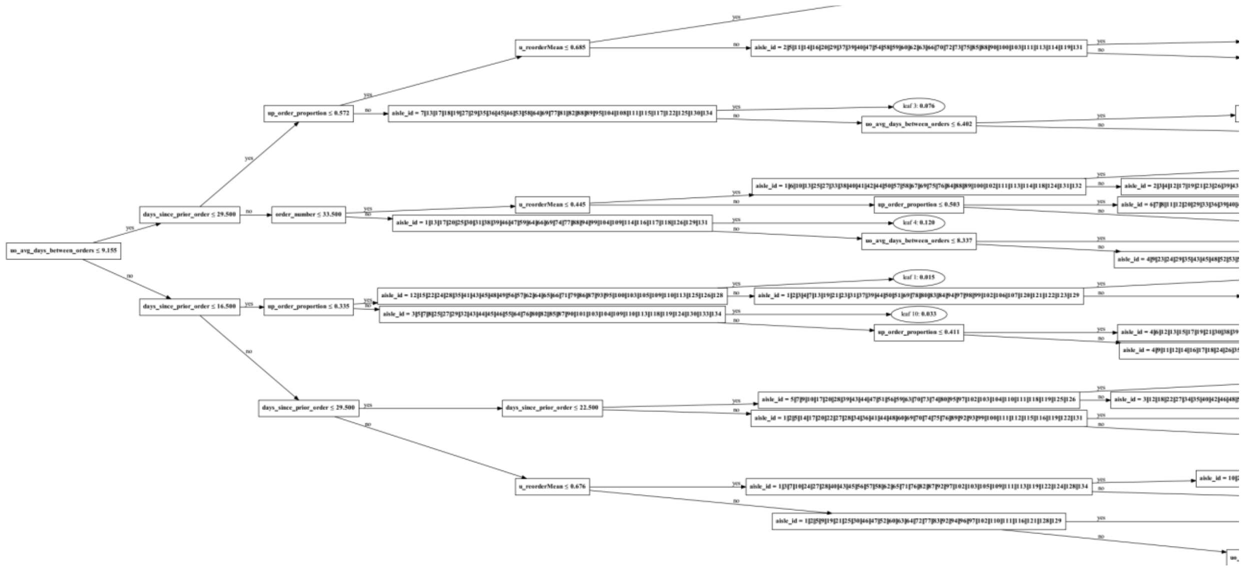


Figure 13 - Tree 23 snapshot

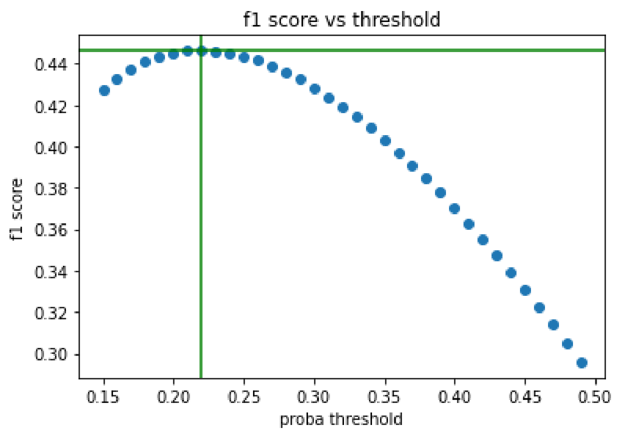


Threshold selection

We choose the threshold that maximizes the F1-score on the training data.

The chart below shows how the F1-score varies by threshold value. The best F1 of 0.446 for the training data occurs using a threshold of 0.22. We use that in calculating the F1 score for the test data.

Figure 14 - Threshold tuning on training data:
Best threshold: 0.22;
F1_score: 0.446



Confusion matrix/Precision/Recall

Confusion matrix

The confusion matrix, presented in the form of a heatmap and a matrix in the two figures below, shows that we predicted only about half the reordered items as reordered (recall = 0.51). Additionally, we had more False positives than True positives (precision = 0.4), due in part to the imbalanced data. Apparently, the not so great prediction capability is one of the reasons retailers are interested in improvements for this problem, since it ends up being our results are not that far off from the competition leaderboard, as we will discuss in the 'Kaggle submission data' section.

Figure 15 - Confusion matrix heatmap

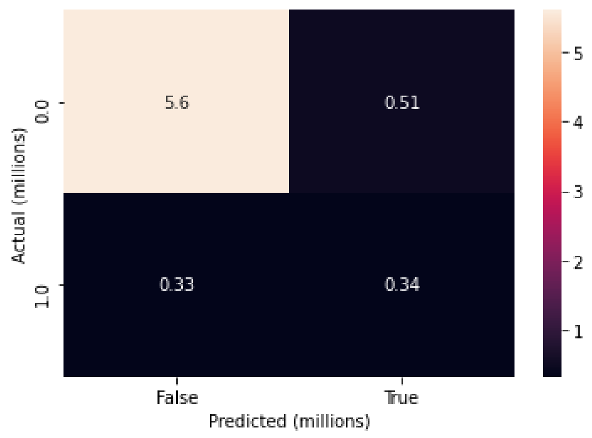


Figure 16 - Confusion matrix - raw data

	Predicted: NO	Predicted: YES
Actual: NO	5,601,014	505,846
Actual: YES	326,745	335,468

Figure 17- Classification report

class	precision	recall	f1-score	support
0	0.94	0.92	0.93	6,106,860
1	0.40	0.51	0.45	662,213

F1-Score: Test data

We use the fitted model to make predictions for the test data, resulting in F1 = 0.4403, which is close to our train data score, somewhat reflecting the effects of splitting from a relatively large dataset.

F1-Score: Kaggle submission data

Finally, we make predictions for the competition submission data. In this case we do not know the expected results; instead they require us to submit a list of the items we predict will be reordered for the 'submission' orders. Kaggle then returns an F1 score.

Our Kaggle submission F1-score is 0.3772. Compared to the best leaderboard score of 0.40914 , we feel this is respectable for our first cut at this model.

Our submission score differs more than we would expect from our 'test' data score. A possible explanation is that we are not calculating the input user/product cross product the same as they are. Another is the 'none' prediction option that is discussed on the Kaggle discussion board along with a competitor-supplied 'kernel' to address. However, the difference is not explored here, and is left for future consideration.

Conclusions and future work

We explored data and developed a baseline light gradient boost model to make product reordering predictions for an on-line grocery service. The model predicts only about half the reordered items, as well as just as many non-ordered items. However, this seems to be a difficult problem, as many competition scores are similar, and leaves much room for future improvement.

Considerations for future work include:

- Additional feature engineering
    - Refining some of the current features.
        - For example, for user-product reorder proportions, we currently account for all orders rather than orders after the item was first purchased.
    - Including additional features
- Implementation of the 'none' prediction feature.