Machine Learning Engineer Nanodegree

Capstone Project
Raj Nikhil Choul
October 26th, 2018

I. Definition

Project Overview

E-commerce is one of those sectors which can benefit immensely from machine learning and various advanced neural network algorithms. This is because not only the data required to build these model easily accessible but it is also relatively easy to deploy them in production to help improve the customer experience. It is no surprise then that some of the biggest e-commerce companies around the world were the pioneers when it comes to building and deploying machine learning models to improve various aspects of shopping on their website.

One of the most impactful application of machine learning on an e-commerce website would be to help find customers what they may need or want without having them to explicitly search for it. This features becomes even more useful for customers of a website, which have a large assortment of products and for website whose customers frequently come back to buy similar products. This not only improves customers' shopping experience but it also builds customer loyalty.

Problem Statement

An year ago, Instacart put out its data related to prior customer purchase in the public domain. It was also a competition on Kaggle. As part of this project I will need to build a model that predicts which previously purchased products will be in a customer's current order. Since the problem requires to predict whether a product will be bought in the current order it can be considered as a classification problem where all the products previously bought by a user have to classified in one of two bins i.e. reorder or no-reorder.

The dataset provided by Instacart provides some good information about purchases by its customers but a lot of the information may be hidden and need to be brought to the foreground by engineering features. From my initial study of the data I did figure out a few of these features that I can create such as when are products most likely bought, duration between purchases and so on. However it wouldn't be as rich in information as when created by a domain expert. For this reason, some of the features have to be inspired by those used in similar models out

there. Once these engineered features are created it would be much easier to build models using few of the many classification algorithms. In the end the model should successfully predict which products will be in a customer's current order based on the factors/features used in developing the model.

Metrics

In suggesting a customer products they may be interested in buying based on their previous purchases the model has to take care of not to present too many products not present too few of them. If the model presents too many products it defeats the whole purpose of building because then the customer is again going through same steps of finding products he/she needs. On the other hand, if the model present too few products the customer may initially find the experience pleasant but then they will need to surf around and find the rest of the product endangering any goodwill the website may have generated so far. For this reason a metric which maintains balance between these two conditions would be the most apt metric to measure the quality of the model. In other words, this means the model should maintain a balance between precision and recall. Precision and Recall tend to trend in opposite direction therefore to maintain a balance between the two without having to measure each individually we use F1 score. Since F1 is the harmonic mean of precision and recall it a good metric to evaluate this model.

$$precision = \frac{True\ Positive}{(True\ Positive + False\ Positive)}$$

$$recall = \frac{True\ Positive}{(True\ Positive + False\ Negative)}$$

$$f1\ score = 2.\frac{(precision.recall)}{(precision+recall)}$$

II. Analysis

Data Exploration

The data for this project was provided by Instacart[1] an year ago. The data was initially part of a Kaggle competition but Instacart has since made the data publicly accessible. The data is divided into 6 datasets. Below are the details of each of the datasets.

1. Aisles: This dataset contains data about category of products placed in this aisle such as Kitchen Supplies, Packaged meats, etc. This dataset has 134 distinct aisle names.
2. Departments: contains data about categories of products such as Beverages, Alcohol, Pets, etc. There are 21 such departments in the data.

3. Products: Products dataset contains names of the products sold on Instacart and also has details of which aisle the product is in and which department the product belongs.
4. Order Products (Prior & Train): These dataset are divided into two, one for prior data and train data. Prior dataset has details of orders prior to the most recent/current order. Train dataset has the same details as that in prior but is provided to train the model on the current order.

   These datasets contain details of order id, IDs of products included in that order, order number in which the product was added to the cart and whether the product was reordered or not represented by 1 or 0 respectively.
5. Orders: This dataset contains details such as order id, user id, eval set i.e. Prior, Train and Test, order number for the user, day of the week the order was placed, hour of the day the order was placed, and number of days it has been since most recent prior order (capped at 30).

All these datasets together provide a history about a customer's prior purchases and the trends in those purchases such as which products a user buys, at what interval are they bought, are there any combinations that are frequently bought together and so on.

The products ordered as part of the test evaluation set are not available in this dataset. Therefore, prior evaluation set is used as train and validation dataset while train evaluation set is used as test data.

The data in the current state is not directly usable to build a model upon because of fragmentation of different aspects of the order. In order to build model, the data has to be transformed so that all the order data is collated in a single dataset. Also, to make data more usable given size of the complete data and limitations in terms of hardware when working such big data there need to be subsets of data created, which lie between data provided by Instacart and the completely collated data. These subsets will facilitate creation of the engineered features. Examples of such subsets are all_data, all_order_details, etc which can be generated with the code provided.

As mentioned above, the size of the data provided by Instacart is significant. To give a brief idea of size of the data below are some of statistics:

Number of customers: 206,209
Number of product: 49,688
Number of aisles: 134
Number of departments: 21
Number of orders: 3,421,083

Final training dataset contains 11,375,135 rows and 3.4gb big, while the test dataset contains 513,146 rows. Considering the size of the data, the quality of data is pretty good with no missing

values and very few outliers. There were a few missing values that were generated in process of creating engineered features. Given their size, which was less than 1% of the training data, it was more economical in terms of time and energy to drop these rows.

A sneak peek at all the datasets:

Aisle Dataset:

	aisle_id	aisle
0	1	prepared soups salads
1	2	specialty cheeses
2	3	energy granola bars
3	4	instant foods
4	5	marinades meat preparation

Department Dataset:

	department_id	department
0	1	frozen
1	2	other
2	3	bakery
3	4	produce
4	5	alcohol

Products Dataset:

	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
3	4	Smart Ones Classic Favorites Mini Rigatoni Wit...	38	1
4	5	Green Chile Anytime Sauce	5	13

Orders Dataset:

	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
3	2254736	1	prior	4	4	7	29.0
4	431534	1	prior	5	4	15	28.0

Products in Orders Dataset (prior & train):

	order_id	product_id	add_to_cart_order	reordered
0	2	33120	1	1
1	2	28985	2	1
2	2	9327	3	0
3	2	45918	4	1
4	2	30035	5	0

User Order Statistics (prior data):

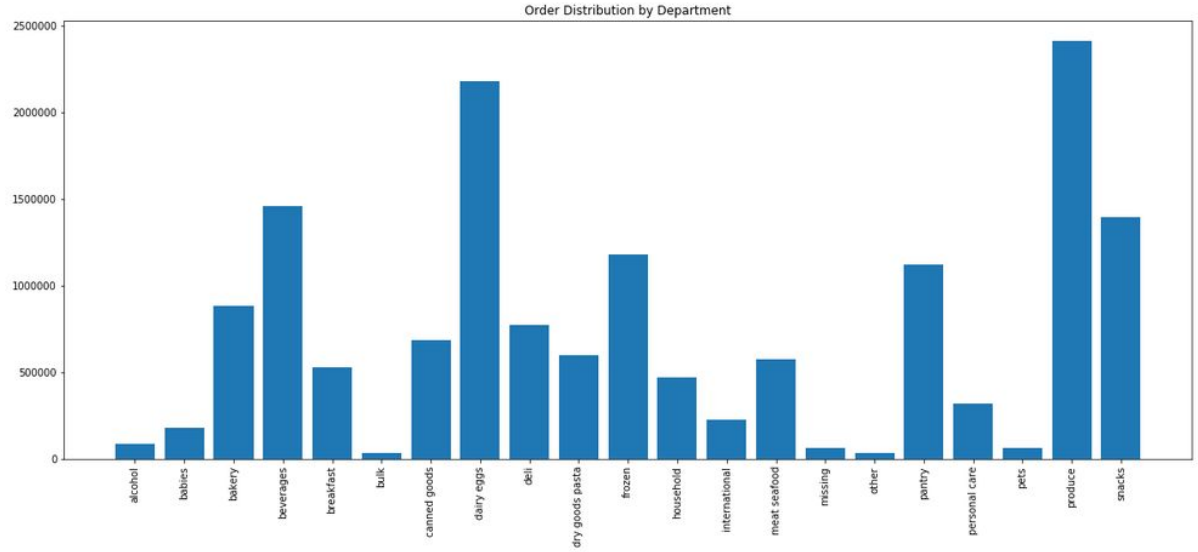
```
count    412418.000000
mean          8.295184
std          13.853167
min           1.000000
25%           1.000000
50%           2.000000
75%           9.000000
max          99.000000
Name: OrdCnt, dtype: float64
```

Exploratory Visualization

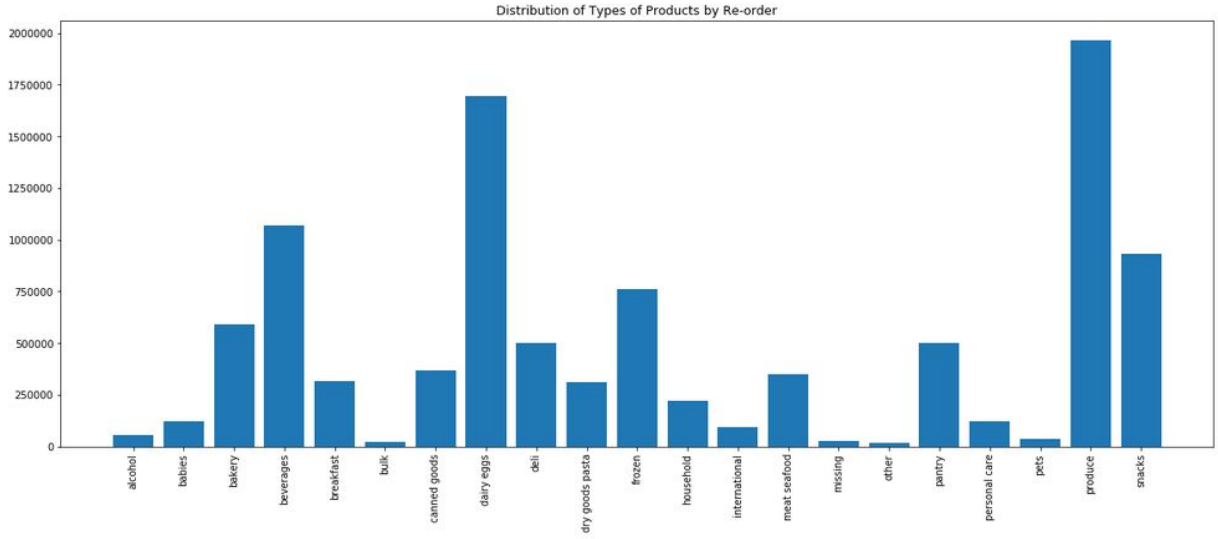
The data exploratory exercise carried out provided important insights into what the characteristics of the data are and which features can be used directly into model building and what features can be created.

Trend between products ordered and re-ordered:

One of the key thing was to determine if there exist some pattern which products are ordered and how that affects their reordering chances. The below chart depicts order distribution by product category (product department).

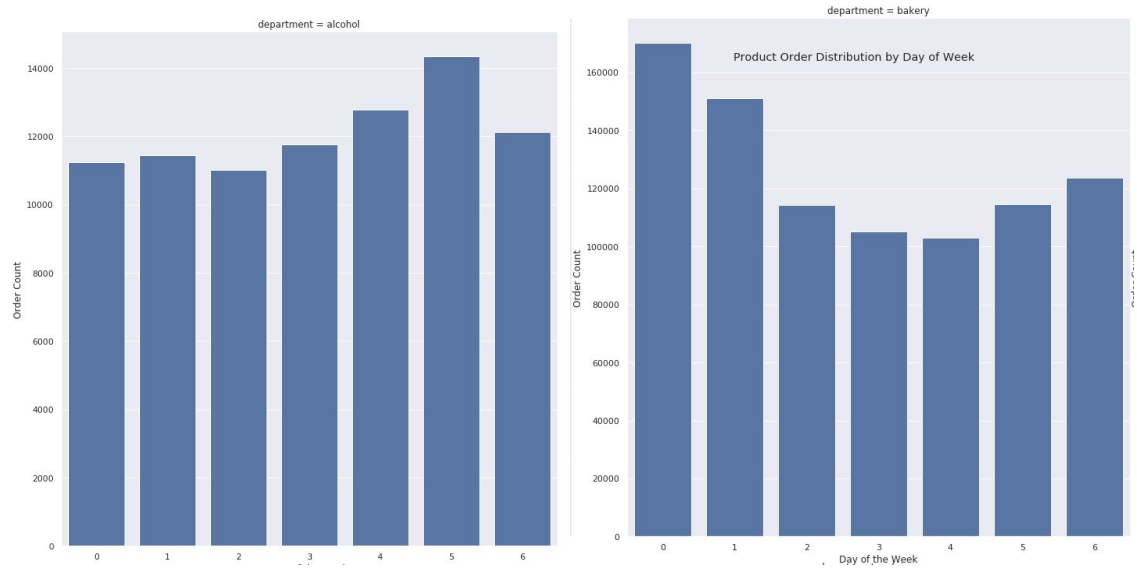


This bar chart shows perishable goods are the most ordered products followed by beverages. The next chart shows reordering of products by product category.



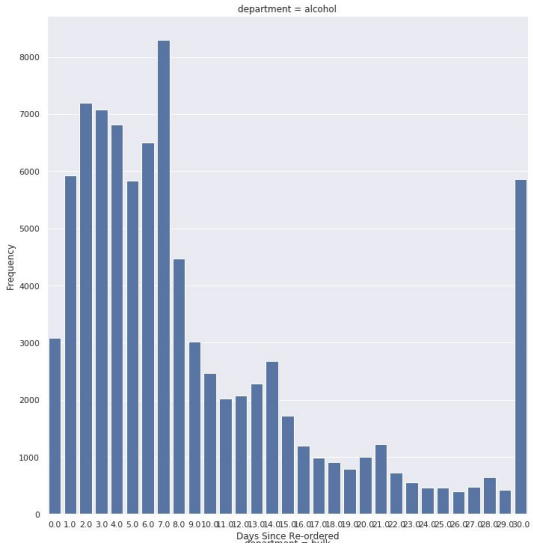
Looking at these two charts it is clear that most people tend to reorder products that they've already ordered before. This led to choosing engineered features such product reorder chance, which along with duration between reorder will prove helpful in determining if any of these products will be in the current order.

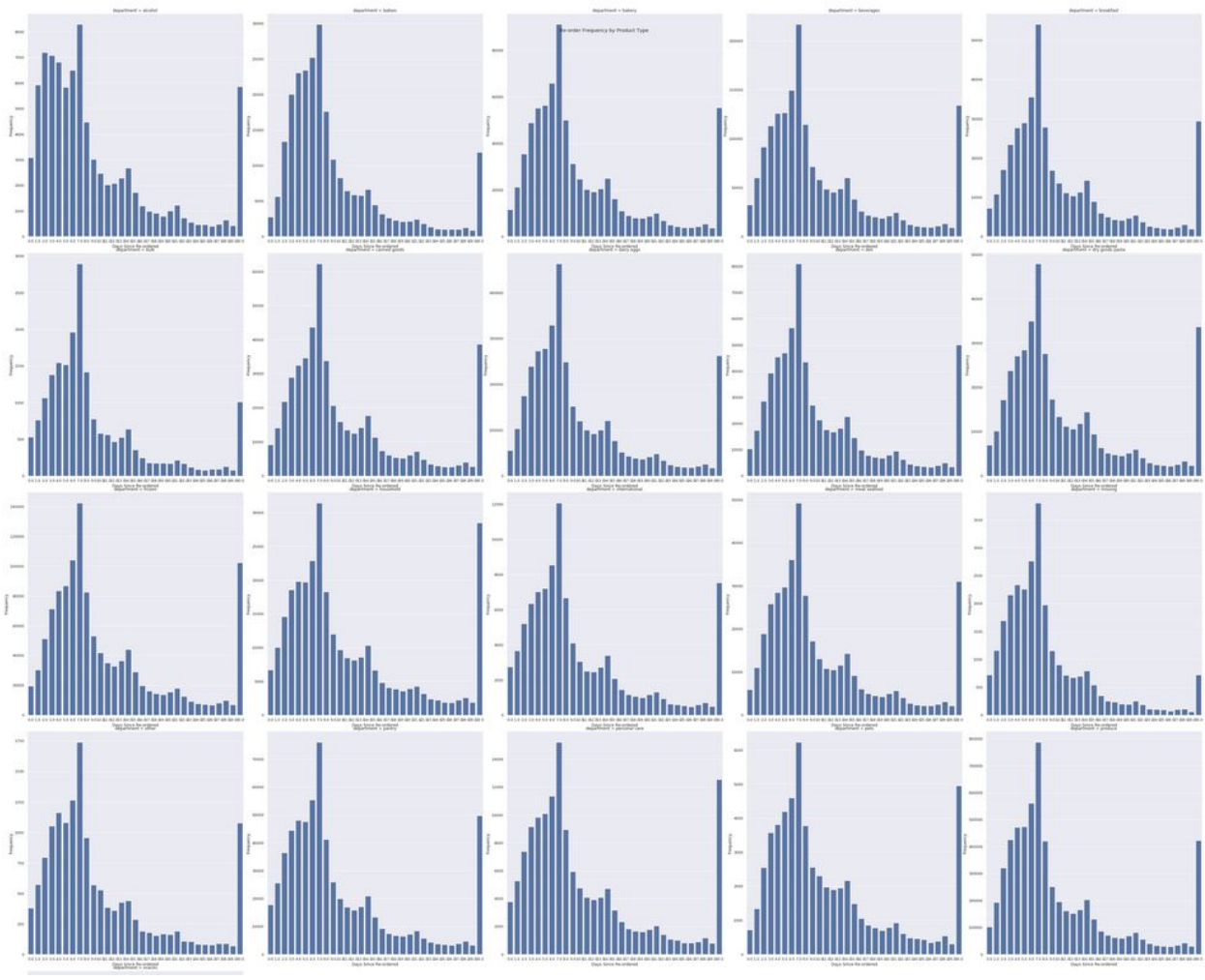
Given how various measures of time could be important, as described above, in determining whether or not a product will be ordered it was necessary to explore data from this angle. Below are some charts which show how orders vary through days of the week and hours of the day.



The above charts represents order trends for alcohol (left) and bakery (right). It is clear the alcohol orders trend higher closer to the weekend while bakery product orders trend higher at the beginning of the week.

Similarly, looking at reorder frequency in terms of days, different product categories have slightly different trend especially alcohol as depicted by below chart. However, most of the product categories peak their reorder frequency at 7 days as can be seen in the second chart below.





Algorithm and Techniques:

The problem at hand in this project is to predict the product(s) a customer will order in the current order given their prior purchase history. Therefore, this is a multi-class classification problem.

There are several variables that may affect product re-order, also, these variables may to varying degrees of affect based on the type of the product. For this reason the algorithms chosen should be able to handle complex trends, in other words, the classification line may not linear. Given these factors, following algorithms should be apt for this problem.

1. Support Vector Machines
2. Random Forest Classifier
3. AdaBoost Classifier

Support Vector Machine:

Support Vector Machine allows for a higher degree classification line, in fact, it doesn't even need a single line. This algorithm is sophisticated enough to handle complex classifications such as encapsulating a class of objects or even classify same class objects at the extreme ends of a distribution. The significant advantage of SVM is that it not only classifies the data but does so with a line that has the maximum distance from the points of different classes. It does so with an error that takes into consideration both classification error and margin error.

$$Error = Classification\ Error + Margin\ Error$$

Along with the error SVM provides for a way to how strictly error should be penalised with the help of C parameter, which is multiplied to Classification error.

$$Error = C * Classification\ Error + Margin\ Error$$

Depending on the requirement C parameter can be changed to tolerate classification or margin error in the classification. Given these features, SVM is suitable for classification of products in the next order as it maximizes the distances between products in the order and not in the order and yet provide a way to tweak the algorithm to give decent classification results without negatively impacting user experience.

Also, SVM can build classification model that require classification line of a higher degree. This is done through the Kernel trick with Kernels such as Polynomial and Radial Basis Functions (RBF). With high complexity in the Instacart data this feature can prove useful in building an equally complex but more accurate classification model.

This way SVM algorithm is suitable in maintaining high user experience levels by being slightly more conservative and suggesting products to customers which have pretty decent chance of being bought in the next order.

Random Forest Classifier:

Random Forest Classifier is an ensemble method, where in it creates several classification trees, all of which have subtle differences between them in that they use different variables to classify, and takes votes from all these trees as to what the predicted value should be for a given case. The resultant classification will be the one with most votes for the given case. This feature helps when the problem is complex and a single decision tree is unable to feasibly accommodate this complexity. Also, even if a single tree is able to classify the data it is most likely to overfit. This means the results on any unseen data will be poor.

The final dataset used to classify products in the next order has several variables, 190 to be precise, which may affect which products will be bought in the next order and their value may be

different for different products. For example, the algorithm builds 3 trees with following structure and results:

1. Tree 1 takes into consideration variables ‘SucceedingOrdCnt’, ‘user_order_size-min’ and ‘department_id_13.0’ and results in products in next order as product 1, 15 & 198

2. Tree 2 takes into consideration variables ‘order_hour_of_day’, ‘user_order_size-min’ and ‘reordered’ and results in products in next order as product 1 & 16

3. Tree 3 takes into consideration variables ‘SucceedingOrdCnt’, ‘user_order_size-mean’, ‘possibility’ and ‘total_buy_ratio’ and results in products in next order as product 1, 15 & 198

The final result for this case would be products 1, 15 & 198 per the voting by individual trees.

Given these, Random Forest Classifier is a good algorithm for this problem.

AdaBoost Classifier:

AdaBoost Classifier is an ensemble method too but instead of taking votes on target variable value prediction, this algorithm builds several classification models and combines them into one final model to predict the final value. These individual models are known as weak learners and all of them together make a strong learner. Since the final model i.e. strong learner is a combination of weak learners, the strong learner is therefore boosted by weak learners. This is why it is called boosting.

Each individual weak learner classifies some of the points correctly and the rest incorrectly. The strong learner is derived by combining these weak learners. However, before these weak learners are combined they have to be weighted based on how good the model of each weak learner is because we want to trust classification of better models more than the worse models. This is done based on the accuracy of the model or in other words how many points were correctly classified by a model. The weight of each model is determined by below formula:

$$weight = ln(\frac{accuracy}{1-accuracy})$$

(or)

$$weight = ln(\frac{\#\ accurate}{\#\ inaccurate})$$

The weights for these models are then added to derive the strong learner. The weight for True class (or product present in this case) are positive and negative for False. If the total weights for a case are positive then the case is True (or the product is present in the next order) and if the

total of the weights is negative then the case is negative (or the product is not present in the next order).

These features of Adaboost make it a candidate algorithm for classifying products in the current order because of the complexity of the data, which may make it difficult for one learner to classify products difficult thereby resulting in poor model.

Benchmark

One of the models available from the competition to benchmark my algorithm is the model that was placed second[2]. The F1 score of this model was 0.4082039. Since the model uses the same metric for evaluation it should be easy to compare the model I develop.

In this model, the author created several engineered features such as grouping customers, products, what kind of products do these customers prefer, how often a product is bought, how long has it been seen being bought previously, etc. The benchmark model uses XGBoost to develop the model for this problem. Since the model was measured with F1 score the reorder probabilities needed to be converted to binary i.e. 1 for reorder and 0 for no-reorder. In order to convert these probabilities to binary a threshold must be set on the probability beyond which a product can be deemed reorder or 1 in a given order.

III. Methodology

Data Preprocessing

The data provided by instacart was broken in order to make it easy for distribution. This meant to make the data usable the first step in data preprocessing was to combine them.

The first set of combination was to collate all product related details i.e. what category a product belongs to and where it is placed. To achieve this, the products tables was combined with aisle and department table. Then the next step was to gather all order related details in a single dataset. The original orders dataset did not have details of products such as the name of the product, what category, etc. without which the data was not intuitive and was difficult to determine what kind features can be engineered. The last step was to gather all data in a single dataset, i.e. what evaluation set the order belonged to, which customer placed the order, how many times a product was added to the cart and whether if the product was a re-order or was being purchased for the first time.

The first two datasets then become a subset of the final dataset. It was not only necessary to go through the first two steps because they were required to create the final dataset but also because the first two datasets made it easier to create transformed or engineered features as they are smaller in size and therefore much quicker.

Following the creation of these dataset, the next steps were to create engineered features. As described at the beginning of the report a domain expert would be an ideal person to help create these features and add color to the data and because of lack of such a resource these engineered features were inspired by other sources based on my understanding of their possible importance. Below is a list of all such features and a description of what information they provide. For ease of understanding, these features are grouped into 4, one which are concerned with the customer, two which are concerned with the products, three which are concerned with the interaction of user with the products and four which are concerned with time

a. User Behavior Features (derived from FE2_UserBehavior.ipynb)

- *reorder_ratio*: mean of how often a user reordered
- *nonReOrd:* no reordering of previous order
- *nonReOrdRatio*: mean of how often a user did not reorder
- *nonReOrdTot*: Total number of non-reorders
- *uniItmTot*: Total unique items by user
- *uniItmTot_Ratio*: ratio of unique items in an order to the number of orders
- *ordItemTot*: Total items ordered by user
- *ordItemTot_Ratio*: ratio of reordered items in an order to the number of orders
- *AvgReOrdDuration*: average duration (in days) between reorders for a user
- *DOW_Mode*: Most Frequent Order Day of week For Users
- *user_order_size-min*: minimum user order size in terms of number of products
- *user_order_size-max*: maximum user order size in terms of number of products
- *user_order_size-median*: median user order size in terms of number of products
- *user_order_size-mean*: mean of user order size in terms of number of products
- *user_order_size-std*: standard deviation of user order size in terms of number of products
- days_since_first_order: number days since first order of a user (FE1_DataPreperation.ipynb)

b. Product Characteristics (derived from FE3_ProductCharecteristics.ipynb)

- *item_hour_cnt*: units of product ordered by hour of the day
- *itemHrDist*: ratio of units of product ordered by hour of the day to total units ordered
- *unique_item_hour_cnt*: units of unique product ordered by hour of the day
- *unqItemHrDist*: ratio of units of unique product ordered by hour of the day to total units ordered
- *item_dow_cnt*: units of product ordered by day of week
- *itemDowDist*: ratio of units of product ordered by day of week to total units ordered
- *unique_item_dow_cnt*: units of unique product ordered by day of week

- *unqItemDowDist*: ratio of units of unique product ordered by day of week to total units ordered
- *prodComb_mean*: mean of times set of products Frequently Bought Together
- *prodComb_min*: minimum of times set of products Frequently Bought Together
- *prodComb_max*: maximum of times set of products Frequently Bought Together
- *prodComb_std*: standard deviation of times set of products Frequently Bought Together

c. Customer Product Interaction (FE4_CustomerProductInteraction.ipynb)

- *possibility*: Possibility of ordering a Product
- *useritem_cooccur-min-min*: difference of minimum user order size and minimum frequency of a product for a user.
- *useritem_cooccur-max-min*: difference of maximum user order size and minimum frequency of a product for a user
- *useritem_cooccur-max-max*: difference of maximum user order size and maximum frequency of a product for a user

d. Time and Day of Order by Product (FE5_Day&Time_Trend.ipynb)

- *dow_order_count*: Count of orders by day of week
- *dow_item_cnt*: Count of items by day of week
- *dow_rank_diff*: difference in rank of dow_order_count and dow_item_cnt
- *hour_order_count*: Count of orders by hour of the day
- *hour_item_count*: Count of items by hour of the day
- *hour_rank_diff*: difference in rank of hour_order_count and hour_item_count

Implementation

After building all the transformed features and collating them into two datasets i.e. train and test work on developing the model commenced. In this final stage few last operations are required to be performed on the data in order to use it to build the model. These operations were not performed so far because they were required during the data pre-processing stage.

1. Missing Values: missing values had to be taken care. A few missing values were created when joining tables as not all values were available in the right tables when left joining them with the table on the left. Such rows were less than 1% of the total data size and with close 11 million rows in this final dataset, it was more than enough data to drop them. It would be easier to develop a separate model for them as those records for non reorders.

   In addition to missing values there were are few rows with infinity value. There were created while building engineered features which had division operations involved in

them. As these values are not able to be processed by python replacing them with a single value made sense. So there were filled with zero.

2. Scaling: Decision trees are not really impacted by the value of the numeric variables as a way check them is built into the algorithms by design. This is not the case with Support Vector Machines since it is an algorithm that is based on euclidean distance of data points from the separation line. To avoid the impact various numeric variables which had different units/scales these variables needed to be all brought on the same scale. This was done with the help of min-max scaling which scales the values of a variable between 0 and 1 with the help of below logic[3].

$$X\_std = (X - X.min(axis = 0)) / (X.max(axis = 0) - X.min(axis = 0))$$
$$X\_scaled = X\_std * (max - min) + min$$

3. One Hot Code Category Variables: The final dataset contained variables which were numeric in value but should actually be considered as category. An example is ID of product category which is numeric but it actually stands for an object. Also, it is easier for algorithms to process these categorical variables if they are in binary format. In order to do this, all categorical variables, i.e. aisle_id and department_id, were one hot coded.

4. Separate Dependent and Independent Variables: The next step is to separate independent variables and dependent variables into their own dataframes to facilitate easy supply of variables to model building libraries. Here, 'product_train' is the dataframe for dependent variable and 'features_train' is the dataframe for independent variables.

5. Sample the training data: This step is the most time consuming step before finally building the model. The training data at this stage has 11 million rows and 191 columns. This is simply too big a data for the hardware I have access to to process and I frequently ran out of RAM resulting in memory error. The most powerful hardware I had access to is 64gb RAM with Xeon E5 2GHz Dual Core processor. For this reason I had to randomly sample training data and go through multiple model building runs before determining what maximum amount of data can be successfully processed without running out of RAM. After several such runs I discovered I was only able to process 0.5% of data which is 56,876 rows.

6. Determine most suitable algorithm: The above step concluded the any data transformation steps. In this step the goal is to determine which of the three potential algorithms i.e. SVM, Random Forest and AdaBoost, is the most suitable one to best predict the products in the current order.

   To do this a small sample of the data was selected to run through each of the algorithm with comparison of training and testing F1 scores. Also, time for model to learn and

predict was measure along with F1 score. Based on these metrics an algorithm would be selected to build the final model on a larger data derived in step 5.

7. Final Model: In this penultimate step, the model selected as best suitable would be tweak to determine optimum parameters. This is done with the help of GridSearchCV library from Scikit Learn. This library will not only run the data through the selected algorithm and cross validate the model results but will also try various combinations of parameters and return the best combination.

8. Predict Products in Test Data: This is final step where in the selected algorithm with best combination of parameters are used to predict the products in orders in the test dataset.

Refinement

The implementation of GridSearchCV to determine was one of the refinement steps implemented in the model building exercise. Doing this exercise programmatically saves significant amount of time and energy, as expected, since a user doesn't have to try combinations of parameters manually and wait until the end of model to see if the results were improved.

Another one of the refinements carried out as part of the project was to manage datatypes of the columns in the dataset. By default whenever a dataset was imported using pandas library, the automated algorithm which tried to detect the column datatype defaulted to high space consuming type such as int64 for integers and float64 for decimal numbers. This consumed a lot of data and given the size of the data (described in step 5 of implementation) the processes failed most of the times due to memory error. In order to get around this, in final data steps of engineering features, every dataset imported was downcasted to unsigned, int8, int16 or int32 and similarly for float. This process reduced the size of the data in memory by 20 to 40% allowing successful completion of the transformation tasks which had failed in previous attempts.

IV. Results

Model Evaluation and Validation

This is a classification and so the candidate algorithms for the problems, as described above, were:

1. Support Vector Machines
2. Random Forest Decision Trees
3. AdaBoost Classifier

It would have been too expensive in terms of time and resources to run each algorithm through a grid search. Instead of doing that, models were built using all three algorithms on a small

subset of data i.e. 1% of the training data with default values for each of the algorithm. I then used F1 score of training and validation dataset along with the time taken to build the model to determine the most suitable model. The idea was, using same dataset with default parameters would avoid giving any advantage to a particular model and should act as a good indicator for suitability of algorithm for the problem at hand

Below are the results of this initial model building exercise:

	SVC	**Random Forest**	**AdaBoost**
Train Time (Sec)	46.7256	9.2232	30.4512
Prediction Time (Sec)	8764.9916	27.7419	357.0798
F1 Score Train	0.96	0.9967	0.0067
F1 Test (validation)	0.0067	0.2188	0.007

SVC took extremely long to train and substantially longer to predict (almost 2.5 hours) and this with default parameters. Even with such long training and prediction times, the F1 score don't present a favourable case. It has an extremely high F1 score in training and significantly low score in testing. This tells its results is an overfitting model. Given these metric values the model was discarded as a candidate algorithm.

Random Forest did slightly better compared to SVC. It was the quickest algorithm to return a model both in training and prediction phases. However, F1 scores don't look promising as, similar to SVC, this too seems to result in an overfitting model and can be discarded.

AdaBoost, however, does look promising. It has reasonable training and prediction times but most importantly its F1 scores on training and testing are really close which tells it will behave in a predictable manner on a data it hasn't seen before. Given these results, AdaBoost was chosen as the algorithm to build the final classification model to predict products in the current order.

The next step was to build the final model. The first thing to do as part of this was to determine optimum parameters using grid search. As expected Grid Search CV was extremely resource hungry and time consuming. Despite the reduced size of the data, as described in the sections above, it took about 8 hours for the GridSearchCV to return results for optimum parameters for AdaBoost classification algorithm.

As part of search for optimum parameters, the values declared for parameters were as follows:

n_estimators: [15, 25]

learning_rate: [0.5, 0.8, 1]

Originally, the plan was to test with two base estimators and few more n_estimators and learning_rate. However, hardware limitations forced to cut down search options.

That being said the F1 score for the model with optimum parameters saw an improvement from the model built during candidate algorithm evaluation phase. With even less amount of data i.e. 0.5% of the total training dataset the F1 score improved from 0.0067 to 0.0103551. This can attributed to the parameter values discovered in the grid search.

Using this model, to predict products in test data - a dataset the model had never seen until now - the F1 score was 0.0159992. This is slightly better than training F1 score but still close enough to confirm this is a good model for predicting products in the current order.

Justification

The benchmark model achieved an F1 score of 0.4082039. Model developed as part of this project falls short of this score with an F1 score of 0.0103551.

The difference in the F1 scores for these two model can be attributed to the follow 3 key differences between the models.

1. XGBoost vs AdaBoost: The benchmark model was built using XGBoost, which is a distributed gradient boosting library. This library uses Gradient Boosting Decision Tree (GBDT) algorithm which is also a boosting algorithm but is still different from AdaBoost in the way it derives these boosters or weak learners.

2. Feature Transformations: The variety and count of transformed features used in the benchmark model are much more exhaustive and definitely provide a lot more value compared to fewer features engineered as part of this project giving the benchmark features a significant advantage.

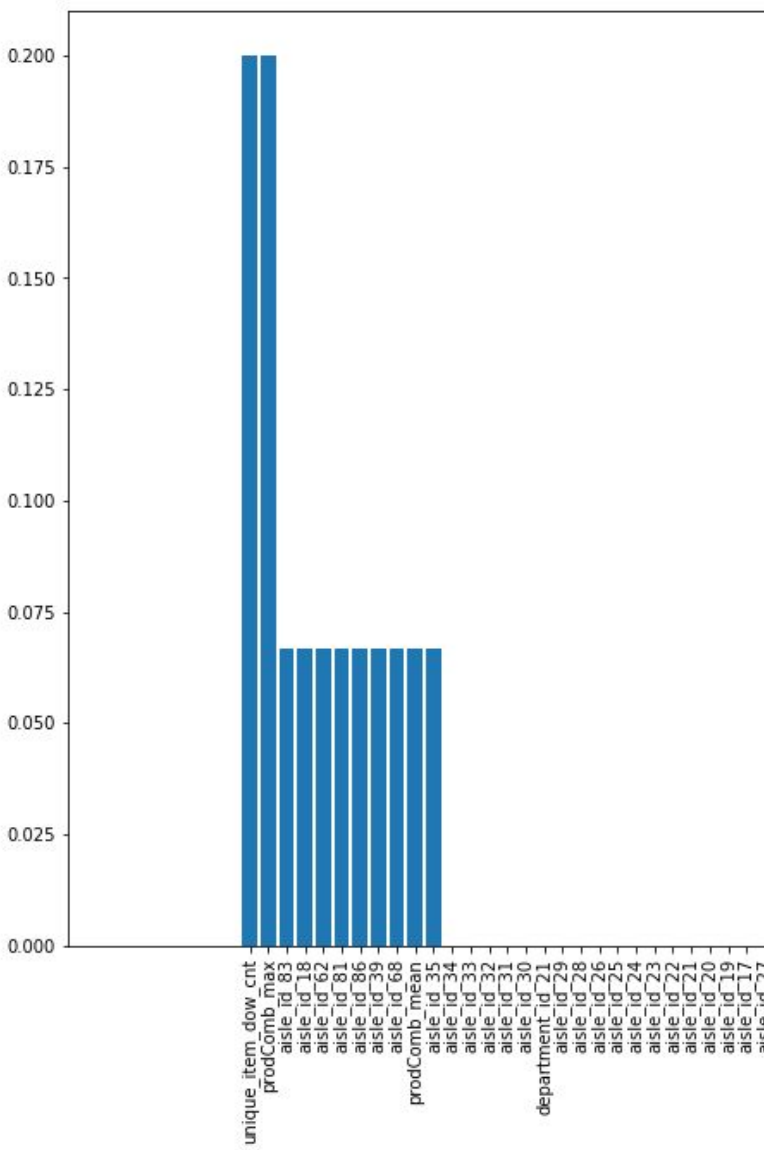
3. Hardware restrictions: The benchmark model was trained on complete training dataset, which may be bigger in terms of rows that the training dataset here as there are more engineered features in there affecting its granularity. In a comment on Kaggle, the author of the benchmark model mentioned that the RAM required for completing model training was close to 300gb. That is more than 3 times the RAM available for this project. In addition to that the benchmark model may have been developed on a distributed system which would mean higher computing power as well. These differences clearly provide an upper hand to the benchmark model.

Given these stark differences in building the two models for predicting products, the results achieved by this model are significant and can definitely be improved significantly with few changes to model, data and hardware.

V. Conclusion

Free Form Visualization

The final dataset that is used to build the model is quite wide, 190 columns to be precise. This makes is pretty interesting to which of those 190 columns are the most useful in classifying which products will be included in the next order. The following visual depicts which are those columns



From this visual it is clear the columns ‘unique_item_dow_cnt’, ‘prodComb_max’, ’aisle_id_83’, ’aisle_id_18’, ’aisle_id_62’, ’aisle_id_81’, ’aisle_id_86’, ’aisle_id_39’, ’aisle_id_68’, ’prodComb_mean’, ’aisle_id_35’

From the above visual it may look like most of the 190 columns a lot of which are engineered features are not useful. But for this model is being developed only with .05% of the data due to hardware limitations. This is why most of the other variables appear not to be important since the sample is pretty small. As the sample size becomes bigger the features which are most important will change. This can also be proved by the fact that many of the current important features are aisle ids, which mean the data may mostly be for products in these few aisles.

Reflection

Executing a modelling project especially on a real life dataset can be both challenging and exhilarating at the same time. The variety and the scale of the Instacart data is immense. When I started with the project, the goal didn’t seem much difficult but along the way I was faced with many hurdles of transforming features, determining to create model specifically for reorders only, ascertaining algorithms to use, handling missing values, etc. Working through these hurdles was quite a learning experience.

The exploratory analysis of various aspects of the data was quite interesting. It was interesting to see how reorder trend was similar to overall reorder trend. This tells most of the products are frequently purchased again. Also, it was fun to know the fact that alcohol reorder increases towards the end of the week while bakery product orders peak at the beginning.

Combining the datasets to create a single dataset which can be used for building the model was the first step of the process to build the model and was also one that would be done pretty quickly. This was fairly easy due to the fact that the data was already pretty clean in terms of missing values. There were some outliers in the data in terms of order counts for users but that data could be useful for determining high value customers and so those records were left in the data.

Things started to get interesting from feature engineering phase onwards. This was a path of many discoveries in terms of capabilities of the pandas and numpy libraries. In the process of efficiently creating these new features I discovered features such as size() - to quantify units instead of iterating and counting with the help counter, rank() - to quickly rank hour of day and day of week instead of sorting and implementing a custom function to do the job. There are other similar examples which in the end gave satisfaction of learning something new and making progress.

Another learning I feel is very important is working with large datasets and managing operations performed on them in ways I hadn’t thought of doing it before. As mentioned earlier, this project experienced severe hardware constraints, which slowed down the entire process. So, in order to

get things moving again I had to dive deeper into technical details and manage datatypes of the variables in the datasets so the operations needed can be completed successfully with the hardware available. Another learning was to discover batch processing the data. Towards the end of creating single dataset 'all_Eng_Features.p' the issue RAM limitations came back to haunt the project and this despite already having downcast most of the columns to the most optimum data types. The next step then was to write out the datasets, clear our ram and read the data in batches and join them and delete the previous batch so as to reclaim memory used by it.

Past these steps, the project entered the final phase of model building. At this stage I was really anxious about my hardware limitations which in the end did turn up and I had to again figure out ways to move things forward. This is when I decided to use only partial amount of data to build the model. After this my plan was to use grid search to determine best model and parameters combination to predict the products but even the partial data was too big and even after several hours of running the grid search there was no output. Then I decided to run an even smaller data sample i.e. 0.5% of data through all the algorithms with default settings. This exercise took time too but did eventually provide results as AdaBoost algorithm having the best performance even though the score compared to benchmark model was considerably lower. After determining AdaBoost as the algorithm to use, I re-ran the data through grid search but this time only for AdaBoost.

In the end I did manage reach the goal of creating a model to predict products a customer would buy in his/her current order albeit not as good as the benchmark model. But even with this model I believe I have achieved a lot in terms of knowledge gained towards becoming a wholesome machine learning professional who can execute a project end-to-end.

Improvement

There are a lot of hurdles I came across during this project, some I was able to overcome others I couldn't. These second set of hurdles provide a lot of room for improvement. One of the first ones which I think would provide the most improvement is to use larger training dataset when building the classification model. For this the best scenario would be to access higher configuration hardware. Other options would be to use a streaming classification algorithm such as SGDClassifier or neural network MLPClassifier. I did consider SGDClassifier because it is a linear model and I am not entirely sure how good a linear model would be for the problem at hand in this project. MLPClassifier again comes down to access to hardware but this may be much easier than accessing a machine with several hundred GBs of RAM.

Another improvement may come from adding more adding more engineered features which are even more complex. There is also possibility in improving predictions of the model by developing separate models for first time purchasers. Even for repeat purchasers there could be separate models based on category or categories of products. These improvements will definitely bring me closer to the benchmark model predictions.

Citations

[1]:*"The Instacart Online Grocery Shopping Dataset 2017", Accessed from https://www.instacart.com/datasets/grocery-shopping-2017 on 09/30/2018*

[2]:http://blog.kaggle.com/2017/09/21/instacart-market-basket-analysis-winners-interview-2nd-place-kazuki-onodera/

[3]: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html