Machine Learning Engineer Nanodegree

Capstone Project

Pradeep Sai Uppula
30 July 2017

I. Definition

Project Overview

Instacart provides a portal to its customers to order products from retailers like Walmart and Costco, and their delivery people would shop for you and deliver products to your door steps. They guarantee 2-hour delivery so time is of the essence for them and able to predict it in advance would be of great advantage to them.

This project is inspired from one of the Kaggle competitions. Instacart has open sourced their orders (over million), The goal of the project would be to predict what the user might add to his cart in the next order. This would be a classification problem where we look at all the items user has bought in the past and try to predict if they would order it again.

Problem Statement

The goal would be to predict next order of each user based on the items ordered before. In the case when user might not buy anything, we should predict empty basket. Let a random user has ordered 10 items before over a period and he kept ordering only within this ten items, we should be able to predict which of these ten items the user might order next.

First important step is to identify what features would impact the reorder probability of product, then based on these features we can develop a feature set and from our database we can exclude the last order placed by user and create a target matrix labelled 1 if product has been ordered last time or 0 if it hasn't been ordered.

Based on these feature vectors and target matrix we can train XGboost classifier and generate output vector. This output vector corresponds to items ordered by the user in past and whether these will be ordered in the next order.

Metrics

Simply giving a percentage of correctly predicted results wouldn't be able to justify the model clearly since if we predict the user orders all the items he has ordered before to be reordered we might end up with decent accuracy but would have a lot of false positives in it.

To overcome this problem, we use F1-score to estimate how accurate the model performs. Precision and Recall would ensure that false positives and false negatives are considered when producing the results, thus helping to build a better accuracy model.

```
F1 = 2 * (precision * recall) / (precision + recall)

Precision = total true positives predicted / (True positives predicted + false positives predicted)

Recall = total true positives predicted / (True positives predicted + false negatives predicted)
```

Higher these values better the result.

II. Analysis

Data Exploration

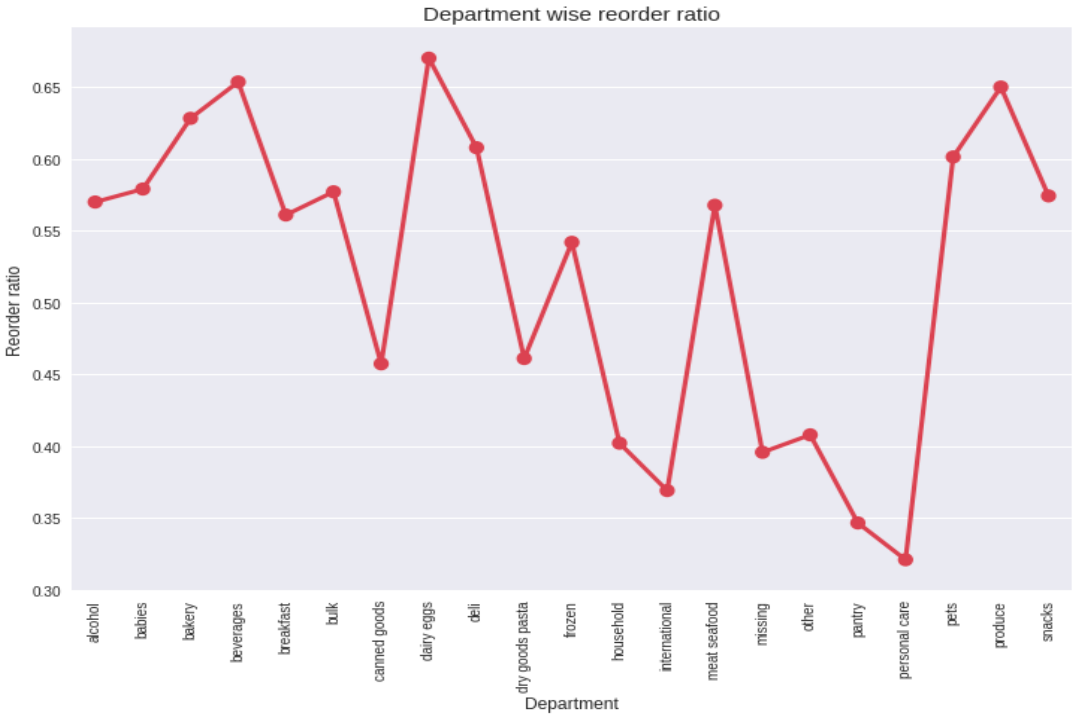
The data sets are obtained from https://www.kaggle.com/c/instacart-market-basket-analysis/data , Instacart has open sourced this data. We have 5 sets of files to train upon and should produce a submission file based on analysis made on these 5 files.

1. Aisles- Contains where in the storage products are located
2. Aisleid-Number value separate each Aisle
3. Aisle - String value describing each Aisle (processed, soups, salads…)
4. Departments - A bit higher level categorization.
5. Departmentid:- Number value to separate each department.
6. Department(frozen, dry):- String value describing each department
7. order_products - Contains information of which product is bought in each order.
8. order_id:- Numerical value indicating order number
9. product_id:- Numerical value describing each product purchased separately
10. add_to_cart_order:- Order in which products are added to cart from the app.
11. reordered :- 1 indicates that the product was bought before.
12. orders: - Mostly self-explanatory
13. products: - Description of each product.

When we consider the 2 files order_products_prior and order_products_train. Both contain identical columns, difference would be that order_products_prior contains all transactions except the last transaction for each user whereas order_products_train contain the last transaction for some users only. We can use these as training set and for all the remaining orders we would be predicting what would be ordered next.

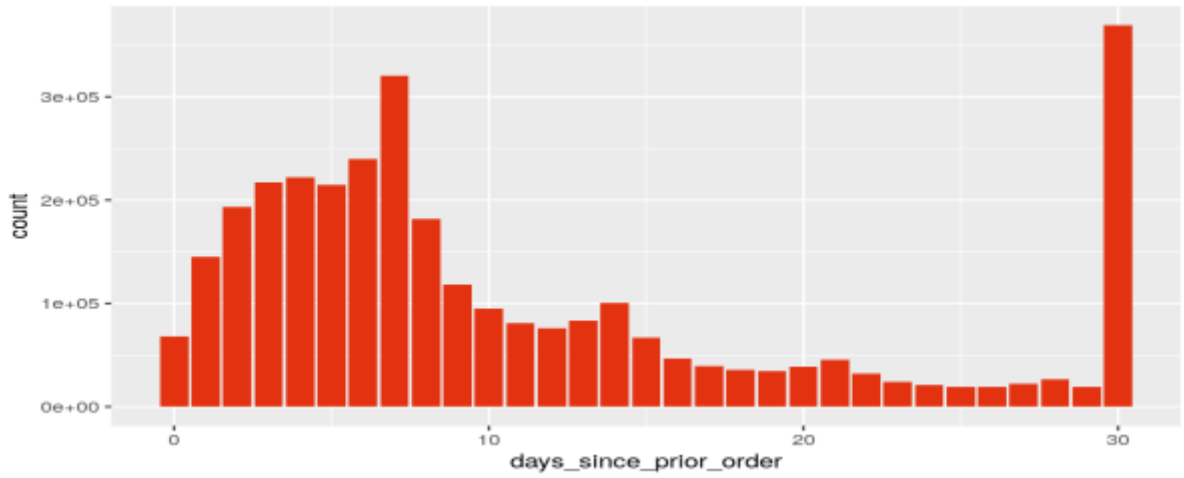
The data set is clearly given, without any missing values except for column of

“ days_since_prior_order” which is “NaN”, Say for the first order days since prior order would not make sense. The features that have been built based on this column calculate mean, so leaving it as “NaN” would be just fine.



There are few interesting patterns, like the one above [4] indicating items from some department are tend to be ordered more compared to items from another department. Like dairy eggs or pet care products, while personal care is less ordered. Reveals how items are ordered overall.

Exploratory Visualization

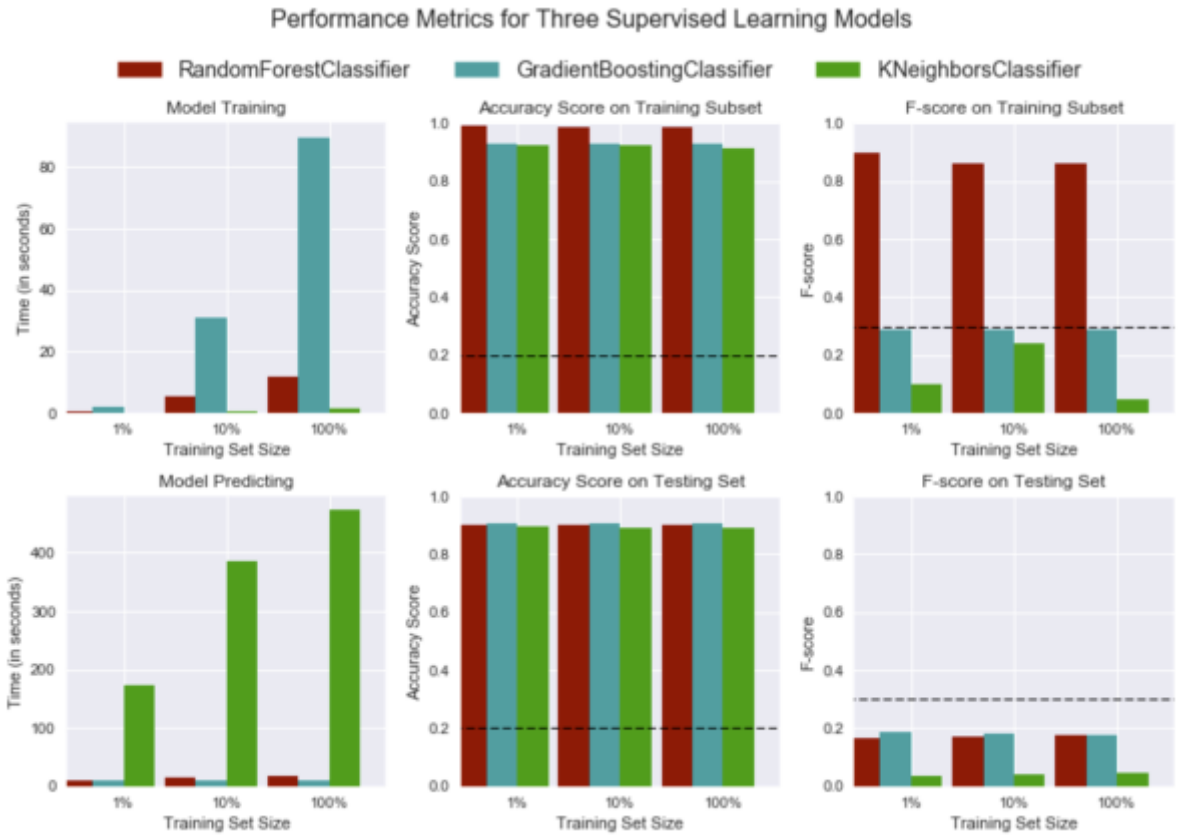


As we can see the users tend to order on a weekly or bi weekly and every month more often, 3-6 days since prior order is also common. The peaks near 7 days and 30 days is quite high. This characteristic along with one discussed above should provide some insight on the way in which users are buying products.

Algorithms and Techniques

Since this is a classification problem, I have tried to use 3 different algorithms and then extracted some initial results. The image in the next page shows comparison of different algorithms (The data set size has been reduced for this comparison to avoid high waiting time). Results indicate that KNeighbours classifier doesn't suit the model very well. However, while comparing Random Forest Classifier and Gradient Boosting Classifier, Although Random Forest performs well on the training subset it performs equally on testing set this might be due to over fitting of the algorithm.

Further for this kind of problem we can afford to take extra training time but want the results to be quickly retrieved, comparing the time taken by these two I would be considering to proceed further with Gradient Boosting Classifier because of quicker predicting time and equal accuracy with Random Forest Classifier.



I used XGboost algorithm [1] which is an extreme gradient boosting technique to speed up the process time. For this all the data is expected to be numerical, we are not using any data that is categorical in our features so it should be good. Feature scaling for tree based approaches doesn't impact the way algorithm would work, thus no form of feature scaling is done.

Considering the huge data set size, it is important that the program runs quickly. Also, the ensemble technique enables the algorithm to add new models to correct the errors of old models until the result is perfect.

The primary goal is to convert the input data into meaningful features. I have first used all the prior orders of the users or history of users to create 21 features, then based on the last order provided in the test set I create a target variable with reordered variable. So my target variable is created from the train orders data or last order of the user. Then the XGboost model is trained based on this data, on all users except test data set users.

Ones the model is trained I have used the input features related to test data set and try to predict which products from the past the user would buy again.

Benchmark

I have picked a benchmark from the leaderboard (https://www.kaggle.com/deeptanshu/frequent-items-baseline-f1-0-3284533) in which products are reorder only if they are present in 10% of total orders, to be more generic the items which have been bought by most people are predicted to be ordered again with this standard.

III. Methodology

Data Preprocessing

The data set doesn't contain any missing values except for one field called "Days_since_prior_order". Only mean is calculated on this field, usually we have large number of orders for each user ranging from 4-100 previous orders, these fields are ignored when calculating that. Since only the first order of the user doesn't contain value it should be fine.

The algorithm I'm using is an advanced implementation of gradient boosting algorithm, feature scaling isn't necessary for this kind of approach. And all the features we are using are already numerical there is no need to convert any categorical data.

Implementation

While starting to implement this project, I didn't have a clear idea on how feature set and target labels would be. Although features seemed obvious at first like how many times products have been ordered before or which aisle each product belongs to, relating it to each customer seemed quite a tedious thing to figure out.

So, I started with less features (only 4 in my first attempt)

1. Reorder ratio (Number of times product has been rerordered)

2. Aisle ID (The aisle from which the product is selected)

3. Add to cart order (The average add to cart order of the product)

4. Avg number of items in the cart (This would be the user property)

And tried to build feature set. Initial thought of creating a list of products for each user and asking the model on whether the product would be selected or not was a slow process, since I had to create a product list for each user and iterate through each product using loops.

Then I figured out groupby [2] would help, and when I grouped users alongside with products they ordered and used basic 4 features described above I was able to get a mean F1 score of 0.25 in the beginning.

Then I figured out it was all about feature engineering next, keeping the flow constant I altered the features and slowly increased each feature to achieve better results.

Consider a single user for whom all this history of products ordered are grouped together, I then calculated each feature of corresponding product, I have grouped features as user features (how often user orders …) and product features (how often these items are reordered) then combined them and used the training set provided to create training features and target.

I have used the built-in library of python to implement the XGboost model. The result of an F1 score was obtained after submitting solution to the Kaggle competition.

Refinement

This project is unique in its way, and I figured that enhancing features has given me better results. My initial algorithm included 4 features which resulted in 0.25 F1 score then I worked on more features like product reorder rate and user reorder rate, and this helped improve the performance.

Some additional features that were added include

1. Avg basket size
2. Avg days since prior order
3. Total items ordered
4. Total distinct items ordered
5. User reorder ratio = sum of reordered products / sum of number of products after first order
6. Reorder probability :- Number of time product is bought again after it has been ordered ones
7. Reorder ratio
8. Reorder time
9. Avg add to cart order

Each of these features had a significant impact on the result and final F1 score was improved to 0.36.

IV. Results

Model Evaluation and Validation

Classifier	KNN	Random Forest	Gradient Boosting	XGBoost	XGBoost improved
F1 score	0.08	.196	0.192	0.35	0.365

Multiple models have been trained and their performance was compared, XGboost after modifying hyper parameters using gridsearchcv achived better results than other implementations. Parameters when binary logistic was used performed better than reg:logistic alongside max_depth which was changed from 5 to 7. Also, a threshold was applied on the probability of predicted product which was set to 0.2 to improve performance. i.e., If a product is predicted to be reordered with a probability of atleast 0.2 the product was added to cart.

The model is trained with 60% of data and 90% of data, results don't make a lot of difference. When model was trained with 60% the mean F1 was 0.343 and when model was trained with 90% the mean F1 was still 0.343 indicating that the algorithm has learnt enough from 60% of data so increasing size of data didn't have any further impact, If the performance must be increased more features need to be added.

The model is robust, adding or removing few outliers to the data wouldn't affect the result.

Justification

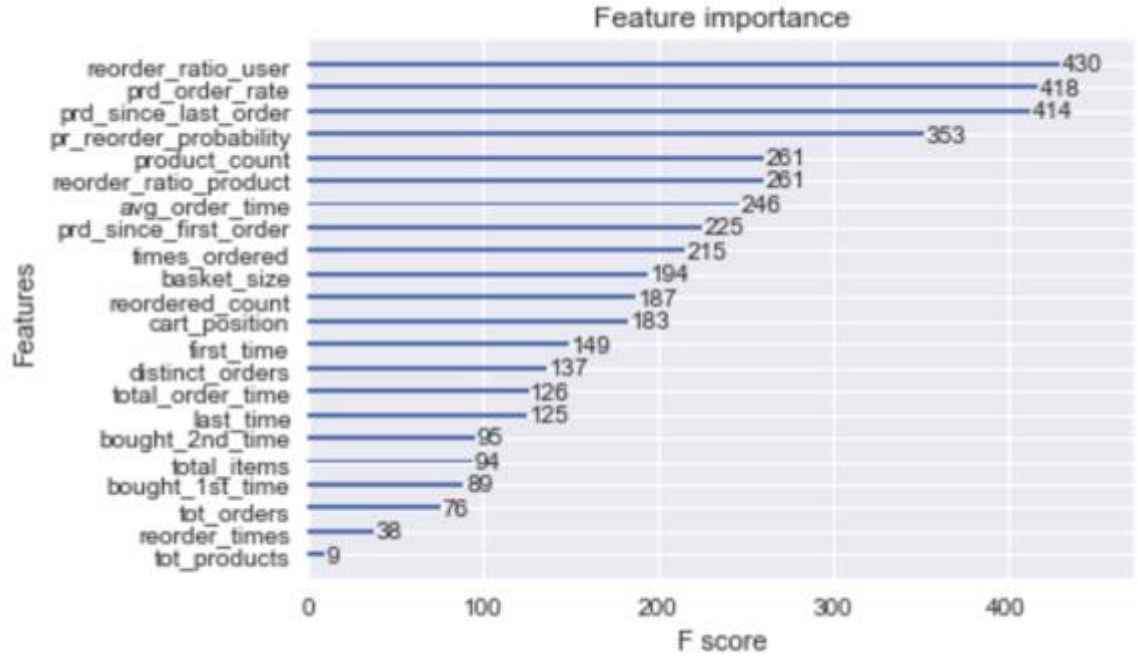
I feel model has performed with improvement from the benchmark model. While the benchmark had an F1 score of 0.32 the added new features provided a mean F1 score close to 0.37. I feel the difference being addition of more features which impacted in correctly classifying more products.

One thing to consider is there are over 50000 products that a user can pick from, when I'm trying to predict the model only uses results from users past to predict if those items would be ordered again. The entire products aren't compared, which would be a drawback since new products are not considered. That would be a different problem on whole something like suggesting user with similar products.

Considering these scenarios, I feel result achieved now would help in solving the problem, Improvements can be further achieved by engineering more complex features.

V. Conclusion

Free-Form Visualization



When we analyze the results from the model we can see “reorder_ratio_user” which signifies the rate of reorders made by individual user has significant impact meaning some users might have tendency to order the same products again and again while others might be experimenting every time the later are the harder group to predict what they might order next.

Consider “prd_order_rate”, this field is related to individual product. Eggs are common products that most of the people would reorder so there are common products or familiar products which would be ordered again most of the times.

As I discussed in Benchmark Section, the bottom features total orders placed or total products purchased by the user doesn’t have significant impact. Since users characteristic might change over larger period of time or depending on different seasons the user might use different products.

Reflection

To summarize the entire process, a list of user’s histories of products is used and combined with features of user alongside features of the individual product, this data is used to train the model to see when products are reordered or not.

The aim of the project is to help maintain a warehouse of items every day from different retailers and quickly (within 2 hours) deliver the products. This project aims at suggesting what items are to be stored in warehouse indirectly. As any machine learning algorithm is only as good as the data, this problem is no different. Considering the large amounts of data that is here It was difficult to figure out how each user would be different from others individually.

Interesting moments in the project were to identify what kind of features differentiated different users. Like user reorder rate, which had a significant differentiating factor.

The process of generating a target variable took a lot of understanding. At the end the solution was simple whether the product would be reordered again or not would be the target and a list of all user's products are maintained in the table. Since data was large using python variables to store variables was very slow, soon realized that data frames were the way to approach these kinds of problems. Using lambda to apply functions on columns of data frame took significantly larger time to compute than creating a new groupby object which performed the task.

The results achieved would certainly benefit the application, but there are more than one Machine Learning techniques that need to be applied on the entire system to make it complete.

Improvement

After working on this project over a month, I feel the improvements in the project could be achieved by developing more complex features, some data that has been hidden is generally the age and gender of the customer. Those features could significantly help in clustering different groups of users, also knowing how many people live in a house, and is it a family or just students living would also help more.

If I had more knowledge of analyzing words, I could use it on product names and see which labels attract the user most, because in their existing implementation they use Spotify's annoy to suggest future products for user. We can see how often user orders from those suggestions and have an estimate of predicting new products other than the user's history.

If my solution was a benchmark, it has a mean F1 score of 0.365. While other Kagglers could achieve 0.4 as their result. There is room for improvement with more feature engineering, in terms of model I have used it looks robust.

References

1. https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
2. https://pandas.pydata.org/pandasdocs/stable/generated/pandas.DataFrame.groupby.html

3. https://tech.instacart.com/3-million-instacart-orders-open-sourced-d40d29ead6f2
4. https://seaborn.pydata.org/tutorial/distributions.html