Capstone Report

1. Define

The project is from kaggle competition [0]. The goal of the project is to find the anomaly in the transactions as fraudulent and non-fraudulent. The competition is brought by Vesta Corporation and its sponsored by the IEEE Computational Intelligence Society (IEEE-CIS) and their researchers who want to improve the accuracy of fraud detection models. Finding transactions which are fraudulent or non-fraudulent is the main scope of the project which is a part of larger research topic of Anomaly Detection.

Although incidences of credit card fraud are limited to about 0.1% of all card transactions, they have resulted in huge financial losses as the fraudulent transactions have been large value transactions. In 1999, out of 12 billion transactions made annually, approximately 10 million—or one out of every 1200 transactions—turned out to be fraudulent. Also, 0.04% (4 out of every 10,000) of all monthly active accounts were fraudulent. Looking at these numbers it has become necessary to deal with fraudulent transactions, these transactions not only impact on the Financial companies but also for the consumer as they have to go through the hassle of recovering these transactions.

One of the ways to find fraudulent transactions real time is to use Anomaly Detections by using Machine Learning models. Anomaly Detections is a process of finding data which are outliers in the given data set. Anomaly Detection can also be used to find other fields where outliers can be detected like in Application server where hackers try to break the system.

Problem Statement:

As defined by Kaggle, “In this competition you are predicting the probability that an online transaction is fraudulent, as denoted by the binary target isFraud.” The submission file contains the probability of fraud for a given transaction instead of a rounded number such as 0 or 1.

Strategy for the solution :

In this project, I would try to solve the problem by using Machine learning. I would first try to find the categories which have a correlation with isFraud binary field.

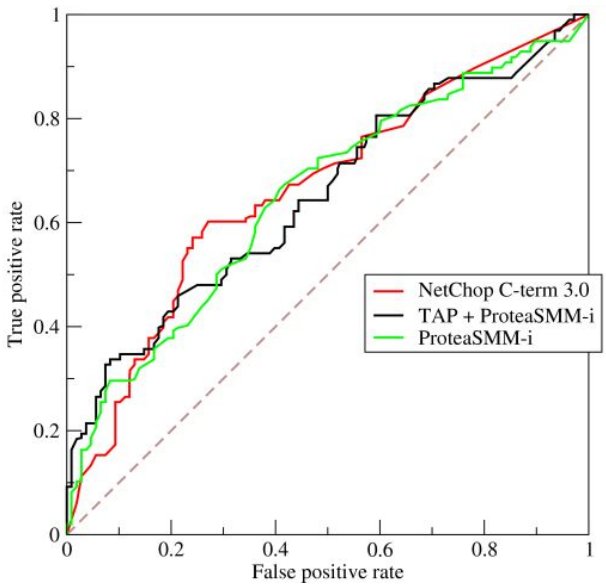
- Will use RandomForestClassifier for my base model, this model will be a Benchmark

model for the final model.

- Will do Feature Engineering for the data.
- **Principal component analysis** (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. Will be using PCA to get correlated variables.
- Finally, will use XGBoost to make the final model.

Metrics:

The submission file will contain the ID for each test transaction and the corresponding probability that the transaction is fraudulent. Kaggle then calculates the area under the ROC curve. The y-axis of the ROC curve is the True Positive Rate (TPR) = TP / (TP + FN) and the x-axis is the False Positive Rate (FPR) = FP / (TN + FP). By examining different cutoff thresholds (when to predict 0 or 1), other than just 0.5, the ROC curve can be created. The curve represents the tradeoff between false positives and false negatives. The optimal AUC score would be 1, implying that the model perfectly classifies transactions.



(Source: https://www.kaggle.com/learn-forum/53782)

Receiver Operating Characteristic (ROC) curves and AUC values are often used to score binary classification models in Kaggle and in papers. A ROC curve plots the performance of a binary classifier under various threshold settings; this is measured by true positive rate and false positive rate. If your classifier predicts “true” more often, it will have more

true positives (good) but also more false positives (bad). If your classifier is more conservative, predicting “true” less often, it will have fewer false positives but fewer true positives as well. The ROC curve is a graphical representation of this tradeoff.

2. Analysis

Data Exploration:

We have been provided 4 files for training and testing. The data is broken into two files identity and transaction, which are joined by TransactionID. Not all transactions have corresponding identity information. Train transaction files contains 590540 data points and identity contains 144233, similarly test data has 506691 and 141907 in transaction and identity files.

Also data has following categorical features in transactions and identity files.

Categorical Features - Transaction

- ProductCD
- card1 - card6
- addr1, addr2
- P_emaildomain
- R_emaildomain
- M1 - M9

Categorical Features - Identity

- DeviceType
- DeviceInfo
- id_12 - id_38

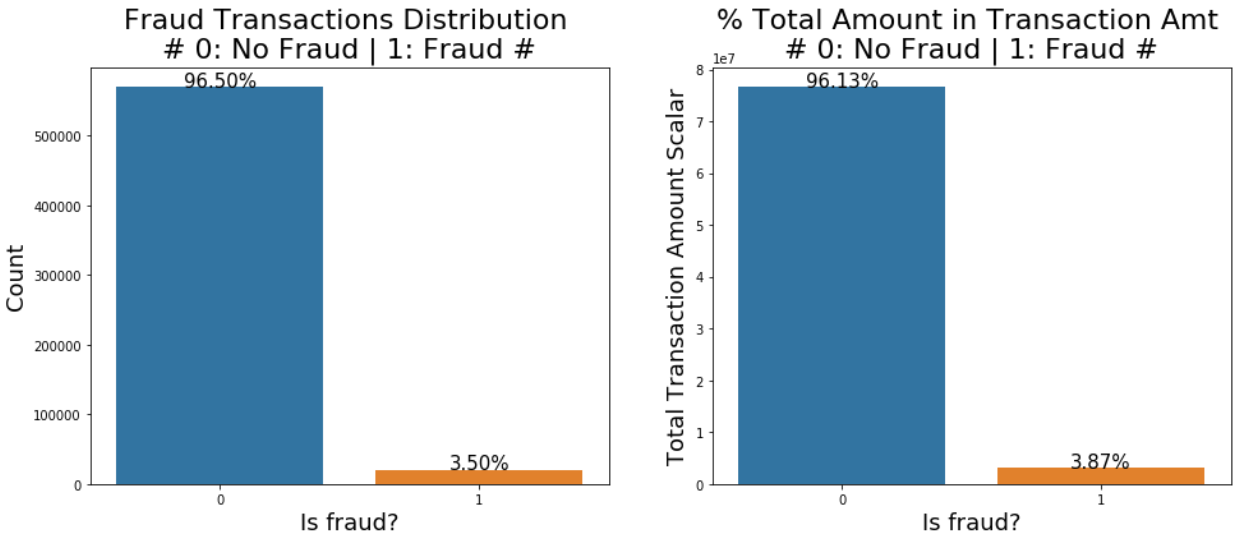
The TransactionDT feature is a timedelta from a given reference datetime (not an actual timestamp).

You can see the top 20 variables with value types and data in below table.

	Name	dtypes	Missing	Uniques	First Value	Second Value	Third Value	Entropy
0	TransactionID	int32	0	590540	2987000	2987001	2987002	19.170000000000002
1	isFraud	int8	0	2	0	0	0	0.220000000000000
2	TransactionDT	int32	0	573349	86400	86401	86469	19.109999999999999
3	TransactionAmt	float16	0	8195	68.5	29	59	8.100000000000000
4	ProductCD	object	0	5	W	W	W	1.280000000000000
5	card1	int16	0	13553	13926	2755	4663	9.970000000000001
6	card2	float16	8933	500	NaN	404	490	6.320000000000000
7	card3	float16	1565	114	150	150	150	0.680000000000000
8	card4	object	1577	4	discover	mastercard	visa	1.090000000000000
9	card5	float16	4259	119	142	102	166	2.660000000000000
10	card6	object	1571	4	credit	credit	debit	0.820000000000000
11	addr1	float16	65706	332	315	325	330	5.060000000000000
12	addr2	float16	65706	74	87	87	87	0.080000000000000
13	dist1	float16	352271	2412	19	NaN	287	6.330000000000000
14	dist2	float16	552913	1699	NaN	NaN	NaN	7.410000000000000
15	P_emaildomain	object	94456	59	NaN	gmail.com	outlook.com	2.680000000000000
16	R_emaildomain	object	453249	60	NaN	NaN	NaN	2.760000000000000
17	C1	float16	0	1495	1	1	1	2.720000000000000
18	C2	float16	0	1167	1	1	1	2.750000000000000
19	C3	float16	0	27	0	0	0	0.040000000000000
20	C4	float16	0	1223	0	0	0	1.120000000000000
21	C5	float16	0	319	0	0	0	2.060000000000000
22	C6	float16	0	1291	1	1	1	2.520000000000000
23	C7	float16	0	1069	0	0	0	0.710000000000000
24	C8	float16	0	1130	0	0	0	1.250000000000000

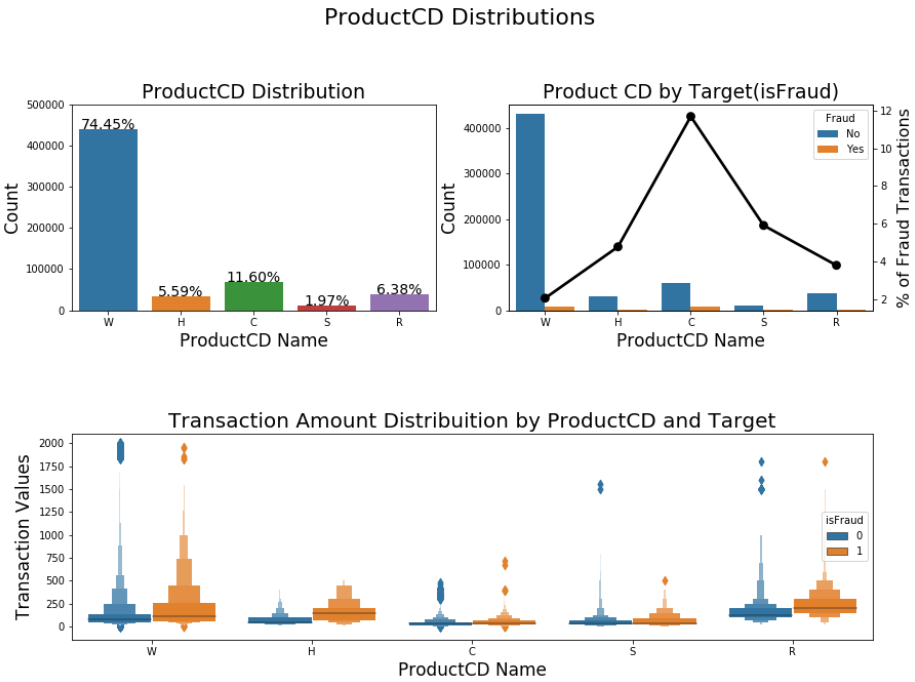
Exploratory Visualization:

We have 394 columns in train_transaction.csv file and 41 columns in train_identity with such an overwhelming amount of feature variables, it seemed more prudent to take a general approach to visualizing the data. My initial step was to examine how many columns are unique or having a missing values in it.



As per above graph I checked the distribution of the amount with fraud and non fraud ratio.

On analyzing ProductCD DDistribution we notice that W, C and R are the most frequent values. We can note that in W, H and R the distribution of Fraud values are slightly higher than the Non-Fraud Transactions



Additionally, there were categorical variables (as defined by the competition sponsor) that had thousands of unique categories. Subsequently, I plotted the class imbalance of the training data set. Of 590,540 training examples, only 20,663 were labelled fraudulent (1). I thought it might be interesting to compare the distributions of fraudulent transactions vs. non-fraud transactions. This might glean some insight into what makes a transaction fraudulent. A feature with a very different distribution when fraudulent provides lots of information. This early analysis may help in evaluating a model after the fact to see if the model decided a feature with different distributions to be important. Here are some interesting insights about the difference between fraudulent and non-fraud transactions in the train set: - TransactionAmt – fraudulent transactions averaged $149 vs. $135 for non-fraud - For the ProductCD variable, 39% of fraud transactions were labelled 'C', vs. 11% for non-fraud transactions.

Card6 – 76% of non-fraud were debit while fraud transactions were almost evenly credit and debit - id_30 – fraud seemed to be less common on MAC OS as opposed to Windows and other operating systems Since I wrote a function to compare the means of numerical features and the categorical distributions of features, I decided to also use this function to compare the train and test sets to see if there were major differences. Here are some interesting insights about the differences between the train and test set: - id_31 – all the browser versions seemed to be later, implying that the test set was collected at a later point in time than the training set. This was also given by the TransactionDT variable and the plot showing that the test set (in orange) was collected later - otherwise the test set distribution did not seem too dissimilar from the train set

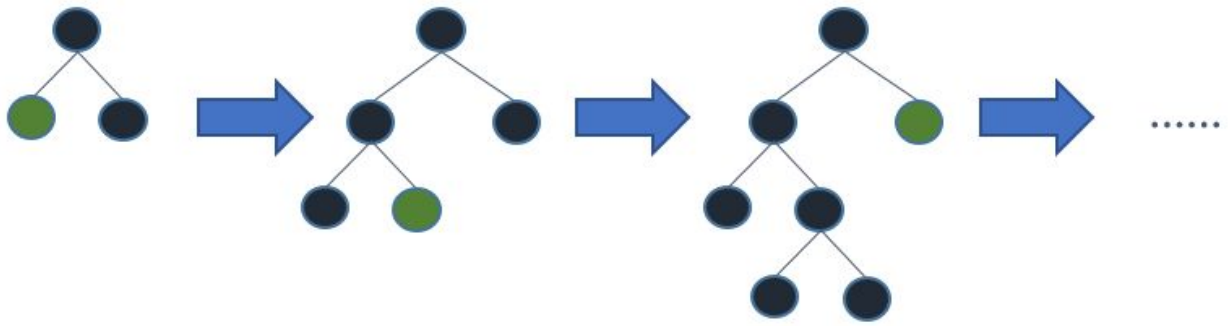
Feature Importance:

Though the code no longer appears in the submission file, I viewed the feature importance output given by the Random Forest model in the first fit. The most important feature seemed to be one of the features starting with 'V' with the importance of around .45. Unfortunately, the most important features were those that started with 'V', 'C' and 'M' – these features that had little to no explanation about them.

Algorithms & Techniques -

1. **Random Forest:** Random forest classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object.
2. **LGBM:** This was the second model I tried, I used basic Feature Engineering and I fit

the model with default parameters. Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks. Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms.



Leaf-wise tree growth

3. **XGBoost:** XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. It uses the Level-wise tree growth

   XGBoost tries to determine the step directly by solving [6][7]

$$\frac{\partial L(y, f^{(m-1)}(x) + f_m(x))}{\partial f_m(x)} = 0$$

   for each x in the data set. By doing second-order Taylor expansion of the loss function around the current estimate f(m-1)(x), we get

$$\begin{aligned} & L(y, f^{(m-1)}(x) + f_m(x)) \\ \approx\ & L(y, f^{(m-1)}(x)) + g_m(x) f_m(x) + \frac{1}{2} h_m(x) f_m(x)^2, \end{aligned}$$

   where g_m(x) is the gradient, same as the one in GBM, and h_m(x) is the Hessian (second order derivative) at the current estimate:

$$h_m(x) = \frac{\partial^2 L(Y, f(x))}{\partial f(x)^2}_{f(x)=f^{(m-1)}(x).}$$

Then the loss function can be rewritten as

$$L(f_m) \approx \sum_{i=1}^{n} [g_m(x_i) f_m(x_i) + \frac{1}{2} h_m(x_i) f_m(x_i)^2] + const.$$
$$\propto \sum_{j=1}^{T_m} \sum_{i \in R_{jm}} [g_m(x_i) w_{jm} + \frac{1}{2} h_m(x_i) w_{jm}^2].$$

Letting G_jm represents the sum of gradient in region j and H_jm equals to the sum of hessian in region j, the equation can be rewritten as

$$L(f_m) \propto \sum_{j=1}^{T_m} [G_{jm} w_{jm} + \frac{1}{2} H_{jm} w_{jm}^2].$$

With the fixed learned structure, for each region, it is straightforward to determine the optimal weight :

$$w_{jm} = -\frac{G_{jm}}{H_{jm}}, j = 1, ..., T_m.$$

Plugging it back to the loss function, we get

$$L(f_m) \propto -\frac{1}{2} \sum_{j=1}^{T_m} \frac{G_{jm}^2}{H_{jm}}.$$

The smaller the score is, the better the structure is. Thus, for each split to make, the proxy gain is defined as

$$Gain = \frac{1}{2}[\frac{G_{jmL}^2}{H_{jmL}} + \frac{G_{jmR}^2}{H_{jmR}} - \frac{G_{jm}^2}{H_{jm}}]$$
$$= \frac{1}{2}[\frac{G_{jmL}^2}{H_{jmL}} + \frac{G_{jmR}^2}{H_{jmR}} - \frac{(G_{jmL}+G_{jmR})^2}{H_{jmL}+H_{jmR}}].$$

Well, all deductions above didn't take regularization into consideration. Note that XGBoost provides variety of regularization to improve generalization performance. Taking regularization into consideration, we can rewrite the loss function as

$$L(f_m) \propto \sum_{j=1}^{T_m}[G_{jm}w_{jm} + \frac{1}{2}H_{jm}w_{jm}^2] + \gamma T_m + \frac{1}{2}\lambda\sum_{j=1}^{T_m} w_{jm}^2 + \alpha\sum_{j=1}^{T_m}|w_{jm}|$$
$$= \sum_{j=1}^{T_m}[G_{jm}w_{jm} + \frac{1}{2}(H_{jm}+\lambda)w_{jm}^2 + \alpha|w_{jm}|] + \gamma T_m,$$

where γ is the penalization term on the number of terminal nodes, α and λ are for L1 and L2 regularization respectively. The optimal weight for each region j is calculated as:

$$w_{jm} = \begin{cases} -\frac{G_{jm}+\alpha}{H_{jm}+\lambda} & G_{jm} < -\alpha, \\ -\frac{G_{jm}-\alpha}{H_{jm}+\lambda} & G_{jm} > \alpha, \\ 0 & else. \end{cases}$$

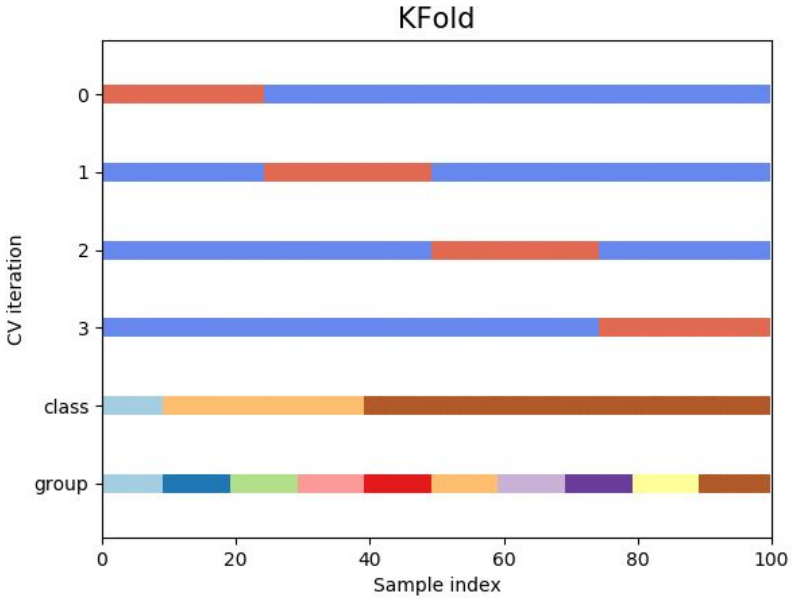
The gain of each split is defined correspondingly:

$$Gain = \frac{1}{2}[\frac{T_\alpha(G_{jmL})^2}{H_{jmL}+\lambda} + \frac{T_\alpha(G_{jmR})^2}{H_{jmR}+\lambda} - \frac{T_\alpha(G_{jm})^2}{H_{jm}+\lambda}] - \gamma$$

$$T_\alpha(G) = \begin{cases} G+\alpha & G < -\alpha, \\ G-\alpha & G > \alpha, \\ 0 & else. \end{cases}$$

4. Model Evaluation

Model Evaluation is an important step in finding a working model. To evaluate the model I used KFold Cross Validation. Usually we split the data set into training and testing sets and use the training set to train the model and testing set to test the model. We then evaluate the model performance based on an error metric to determine the accuracy of the model. This method however, is not very reliable as the accuracy obtained for one test set can be very different to the accuracy obtained for a different test set. To solve this problem I'll be using K-Fold Cross Validation where a given data set is split into a K number of sections/folds where each fold is used as a testing.



https://scikit-learn.org/stable/auto_examples/model_selection/plot_cv_indices.html

Benchmark:

I'll be using Random Forest based as my baseline model. I trained Random forest in my V8 submission and which achieved a score of 0.856823.

3. Preprocessing & Implementation

Handling NaNs:

With LightGBM I replaced empty values with Numpy nan which provided good result but the process took extra time. As XGBoost provides a way to handle NAN and Empty values I replaced those values with -999 and passed configuration to XGBoost to handle those values.

Label Encoding:

Initially I label encoded with selected columns which id12-id38, M1-M9, Email columns, card4-card6, DeviceInfo, ProductCD, addr1 and addr2 which gave me good result with LGBM, while training and testing with PCA and XGBoost I tried encoding all the fields which was type of Object.

Memory Reduction and GPU:

I do not have good hardware with a Graphic card hence I was relying on Kaggle kernel. Working on kaggle kernel is challenge as they have only 13GB Ram with GPU and 16GB Ram without GPU. As XGBoost needs GPU for better performance I had to choose GPU enabled Kernel.

Kaggle kernel stops working once it reaches provided RAM limit hence it was necessary to have better memory management. I used one of the shared function in forum to reduce memory and also deleted unused variables when not needed.

Dealing w/ Class Imbalance:

After reading the XGBoost docs, I found a parameter called scale_pos_weight that is recommended to use when there is class imbalance. The docs recommended setting the parameter to sum(# negative examples) / sum(# positive examples), which would be about 30 in this dataset. I tried a few different numbers near 30 and it did not seem to improve

performance. This leads me to believe XGBoost is fairly robust to class imbalance as compared to other algorithms.

4. Implementation

Hyperparameter Tuning:

In order to find a good set of hyperparameters for the XGBoost model, I tried using hyperopt package to find well performing combinations. However the use of hyperopt package did not improve the baseline XGBoost score of .929

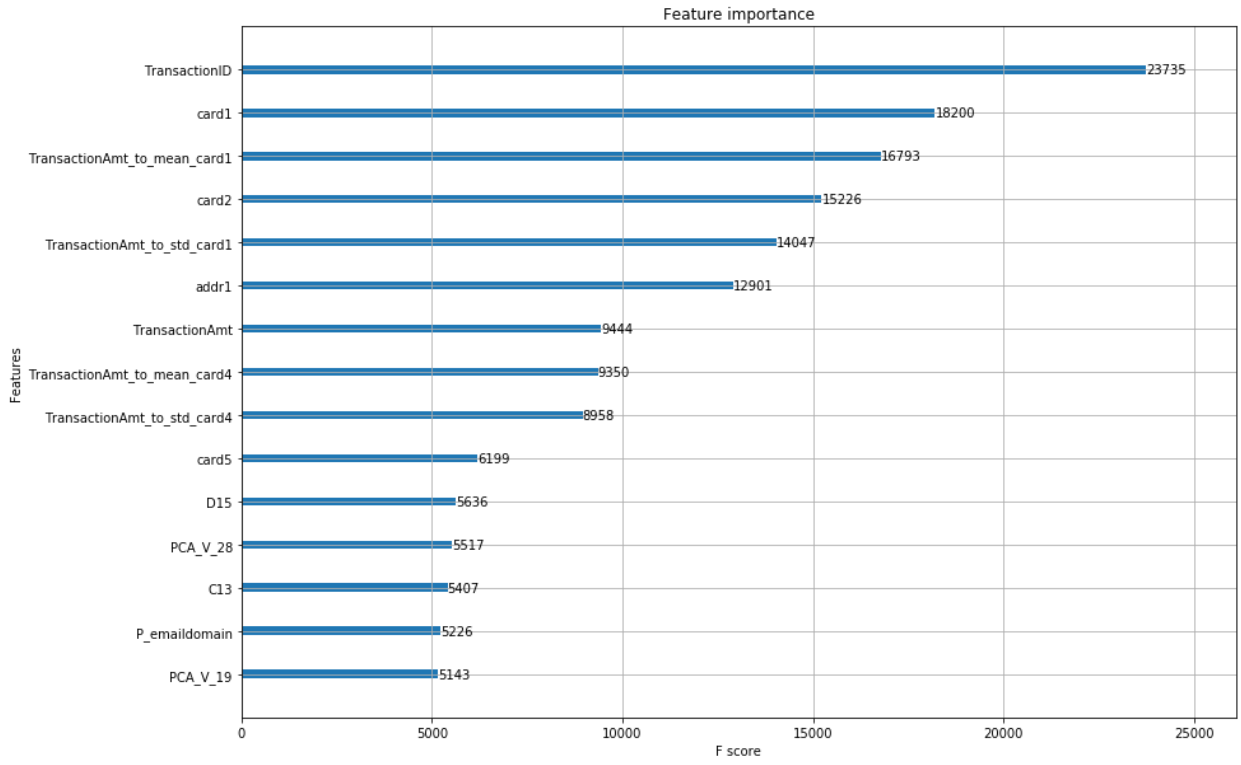
Trial No	1	2	3	4	5
max_depth	10	11	21	22	13
gamma	0.392	0.231	0.42	0.486	0.666
subsample	0.8	0.9	0.2	0.4	0.8
reg_alpha	0.044	0.059	0.339	0.168	0.158
reg_lambda	0.088	0.252	0.218	0.223	0.18
learning_rate	0.172	0.096	0.058	0.096	0.198
num_leaves	90	150	140	70	20
colsample_bytree	0.335	0.757	0.46	0.767	0.355
min_child_samples	140	140	240	170	230
feature_fraction	0.663	0.55	0.724	0.646	0.407
bagging_fraction	0.438	0.883	0.671	0.42	0.839
ROC_AUC Score	0.897	0.908	0.8957	0.9003	0.9008

As per the above table I tried 5 different hyper parameters with the help of hyperopt package. The Trial no 2 gave the best ROC AOC score, with max_depth of 11, gamma value of 0.231 and corresponding parameters as per the above table.

5. Results

I have used Random forest baseline model as my benchmark, few key points of the analysis are as follows.

- In my Random Forest model without any Preprocessing I got a result of 0.856823
- In Version 1 LGBM score I did some preprocessing like Feature engineering and without hyper parameter optimization and here my score was 0.939787
- In Version 6 XGBoost with PCA and Pyper parameter boosting I got public score of 0.922236
- K-Fold accuracy for the final model is 97.66%
- Following chart gives top 15 Important features from XGBoost model.



Conclusions

This was my first project in machine learning which I did on Kaggle, Initially it took time to learn how to use kaggle and different way of using Kernel w/ GPU. I also tried porting data to Google colab but Colab kernel was not as fast as Kaggle in GPU mode.

I learned about XGBoost and LGBM and how to use them on real life data. Below is

final feature importance which I got after applying PCA in XGBoost.

I have realized that having a good EDA and Feature Engineering will surely can improve the performance of the model. There are 2 ways get the Feature Engineering.

- **Manual**: Which I tried with very less success
- **Automated**: We can use Recurrent neural network (RNN) for this task.[4]. I couldn't find open version of the paper hence have kept it for future study in Anomaly Detection.

Project files can be found on my GitHub Repository [2]

Reference:

[0] https://www.kaggle.com/c/ieee-fraud-detection

[1] One_ROC_Curve_and_Cutoff_Analysis.pdf

[2] https://github.com/muke5hy/mlnd-capstone/

[3] https://www.kaggle.com/c/ieee-fraud-detection/discussion/99982

[4] http://bit.ly/2WjGBg8

[5] https://dl.acm.org/citation.cfm?id=3208204&dl=ACM&coll=DL

[6] https://towardsdatascience.com/boosting-algorithm-xgboost-4d9ec0207d

[7] https://arxiv.org/abs/1603.02754