Capstone Report

1. Define

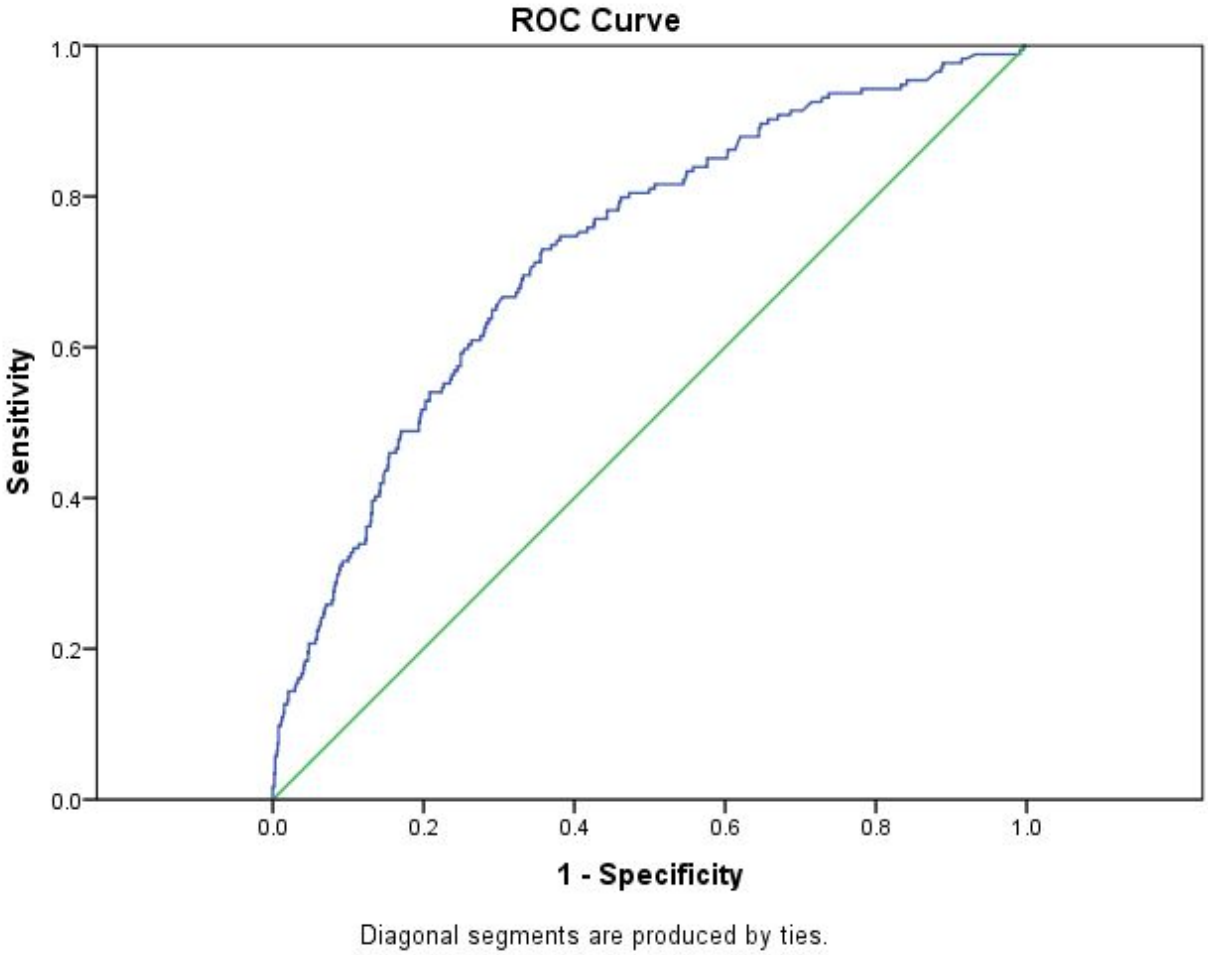
The project is from kaggle competition [0]. The goal of the project is to finnd dthe anomaly in the transactions as fraudulent and non-fraudulent. The competition is brought by Vesta Corporation and its sponsored by the IEEE Computational Intelligence Society (IEEE-CIS) and their researchers who want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

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.

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.theanalysisfactor.com/what-is-an-roc-curve/)

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

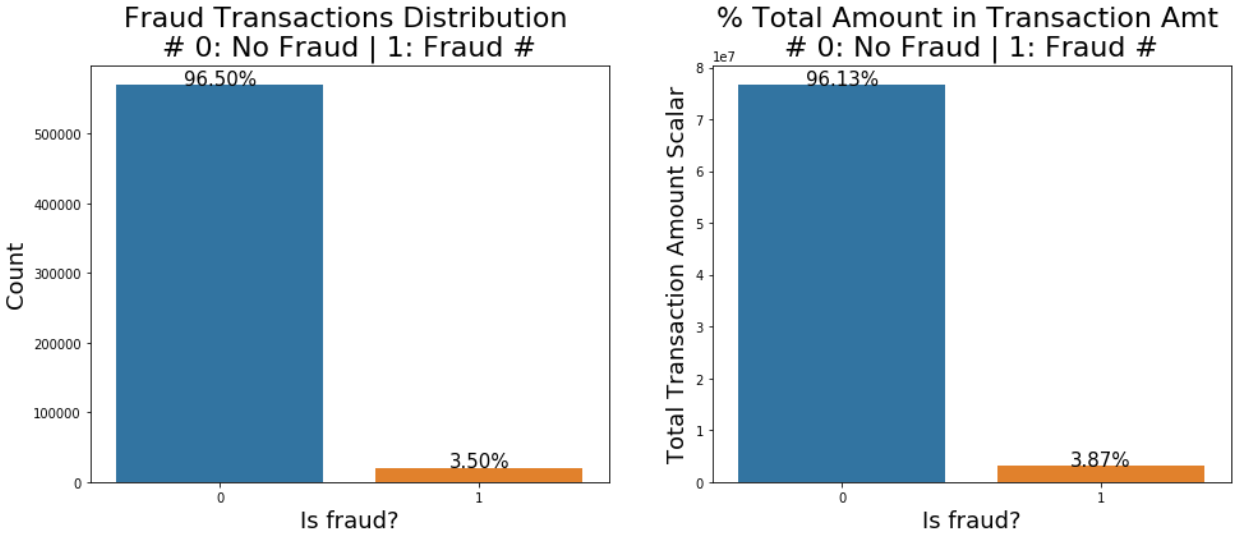
Categorical Features - Identity

- DeviceType
- DeviceInfo
- id_12 - id_38

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

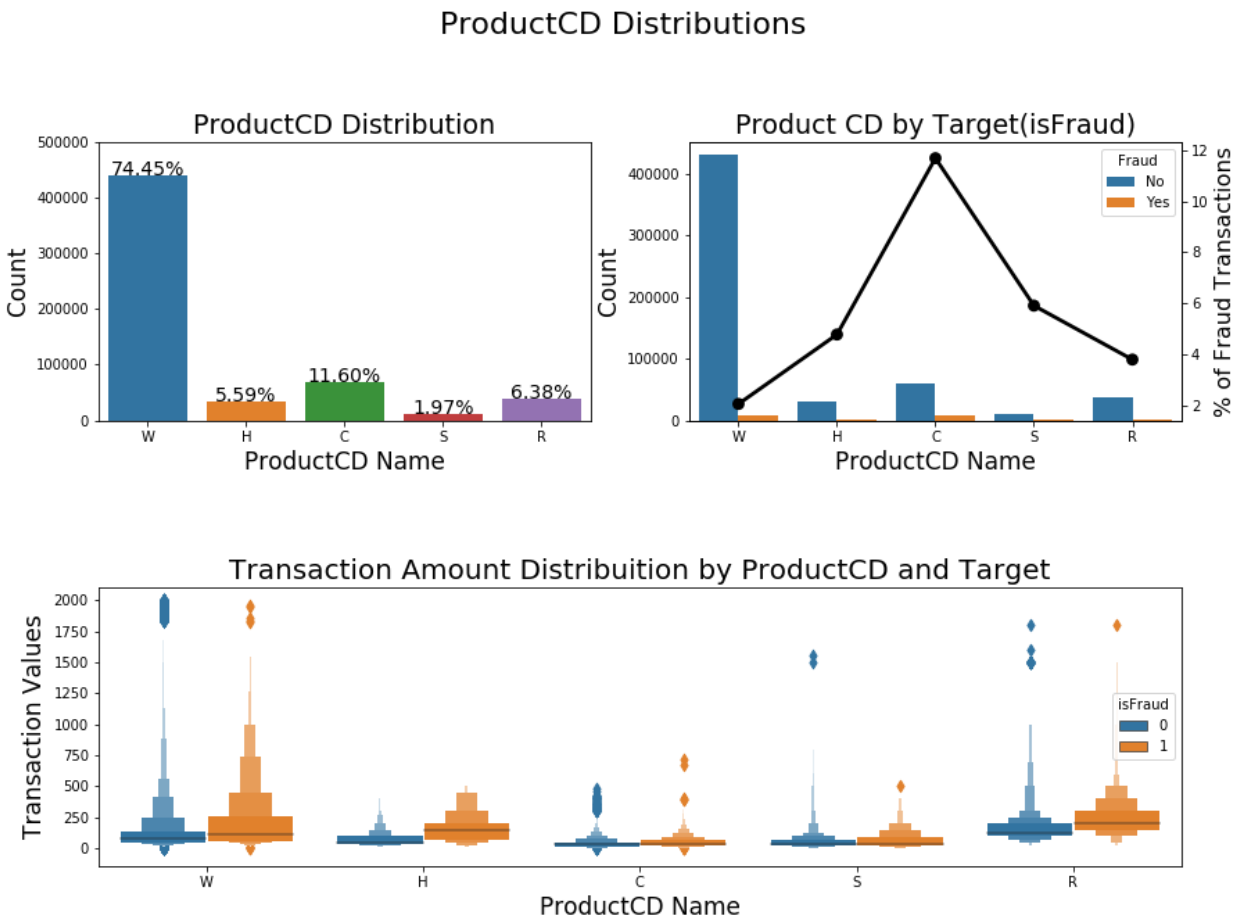
Exploratory Visualization:

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 an 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. **LGBM:** This was the very first model I tried in order to establish a baseline model and score. There was no preprocessing done and I fit the model with default parameters. It achieved a score of 0.93 in v1 of my submission.
2. **XGBoost:** I tried XGBoost with same features as LGBM and I got a score of 0.929, which is comparatively less than what I had gotten with LGBM.
   In my final attempt I used PCA for choosing best parameters but to my surprise the final result was less than my previous XGBoost result. In my 6th V I got 0.922236 score.

Benchmark:

The public leaderboard on Kaggle establishes a reference for evaluating the performance of model. As the competition has ended I couldn't find a way to evaluate my score but as looking at my score I feel I this model would on top 500.

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 with Cross Validation:

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

5. Results •

Public Leaderboard Results:

- In Version 1 LGBM score without hyper parameter optimization public score 0.939787
- In Version 5 XGBoost with same Feature engineering NAN setting as LGBM got public score of 0.943891
- In Version 6 XGBoost with PCA and Pyper parameter boosting got public score of 0.922236

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, I have realized that having a good EDA and Feature Engineering will surely can improve the performance of the model. 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/