IEEE FRAUD DETECTION

(0.931525 ROC AUC SCORE ON KAGGLE)

Group 2

Team Members

Ritu Sharma(rxs180051), Andy Tran(120030),Vishakha Nangia(vxn180007),Mandeep Singh(mxn170019)

Table of Contents

Introduction:

The idea for pursuing this project came from Kaggle competition on IEEE fraud detection. The objective behind the project was to find the best solution for fraud prevention in the financial industry. This competiton was organized by IEEE-CIS (Computational Intelligence) partenered with Vest Corporation which is one the world's leading payment service company.

What the organizers were looking for was a machine learning model that efficiently detects fraudulent transactions and help reduce the loss incurred by businesses across the globe. If some kind of fraud happens with the transaction the company should immediately block the card. The model should avoid making a transaction fraud which is in reality is a legit transaction. The model should provide the probability estimate of a transaction being fraud.

Data Description:

The dataset used for this project was provided by Vesta Corporation. The data is divided into two files identity and transaction, which are joined by TransactionID. There are many transactions in the data with missing identity information. Below is the description on data.

Transaction Table: Transaction data comprised of 590540 examples and 394 variables and was a large dataset to manage. Most of the detailed description of the features were hidden and pairwise directory was not provided due to privacy reasons as the data was real world.

- TransactionID: Transaction ID is the unique ID for each transaction and acted like an identifier for the transactions.
- TransactionDT: Transaction date and time is the date from a given reference datetime (not an actual timestamp)
- TransactionAMT: Transaction amount is the actual payment amount in USD
- ProductCD: Product Code is the product code of the product used for each transaction.
- card1 - card6: These are the variables on card details like payment card information, such as card type, card category, issue bank, country, etc.
- addr: This the address linked to the card.
- dist: distance
- P_emaildomain and R_emaildomain: These are the email domains of the purchaser and recipient.
- C1-C14: These are set of counter variables which are counting values, such as how many addresses are found to be associated with the payment card, etc. The actual meaning of these variables was masked for the purpose of privacy.
- D1-D15: These are the set of variables represent time delta, such as days between previous transaction, weeks between transaction, etc.
- M1-M9: These are match variables, that matches names such as names on card and address, etc.
- Vxxx: Vesta engineered rich features, including ranking, counting, and other entity relations.

Identity Table : Identity data comprised to identity information spanned across 41 variables and 144233 rows.

- Network connection information : Network connection information comprised of information such as IP address, ISP and proxy details of the device that was used for the purpose of transaction.
- digital signature: Digital signatures of the transaction were captured in the features like UA/browser/os/version, etc. associated with transactions.
- TransactionID: Transaction is the common feature shared across both transaction and identity dataset. This feature is used for joining the two datasets.
- DeviceType: Device type contained information on whether the device is a tablet or macbook or any other mobile device.

- DeviceInfo: Device info consisted of other information on the device that was used for the transaction like the model of the device, version of the device.
- id12 - id38: Other identity features whose description was masked.

Challenges:

1. Dataset was huge with 394 features in transaction dataset and 41 features in identity dataset.
2. Identity was hidden for most of the features due to privacy reasons and it was difficult to interpret the features and make sense out of them by just looking at them. This required extensive exploratory data analysis to make sense out of features. Extensive feature engineering was performed and new features were created from old features without losing any information.
3. There were many features and examples with missing values and dropping them was affecting the accuracy and recall scores. This was solved using parameter optimization and 0 imputation.

Exploratory Data Analysis:

Exploratory data analysis is essential for every project. It helps get familiar to the data and make better use of the data. It becomes even more important when the person working on the problem does not have a considerable amount of domain knowledge. Exploratory data helped us get a better understanding of the data as data description was missing for most of the features that made this project very challenging to accomplish.

Data Splitting:

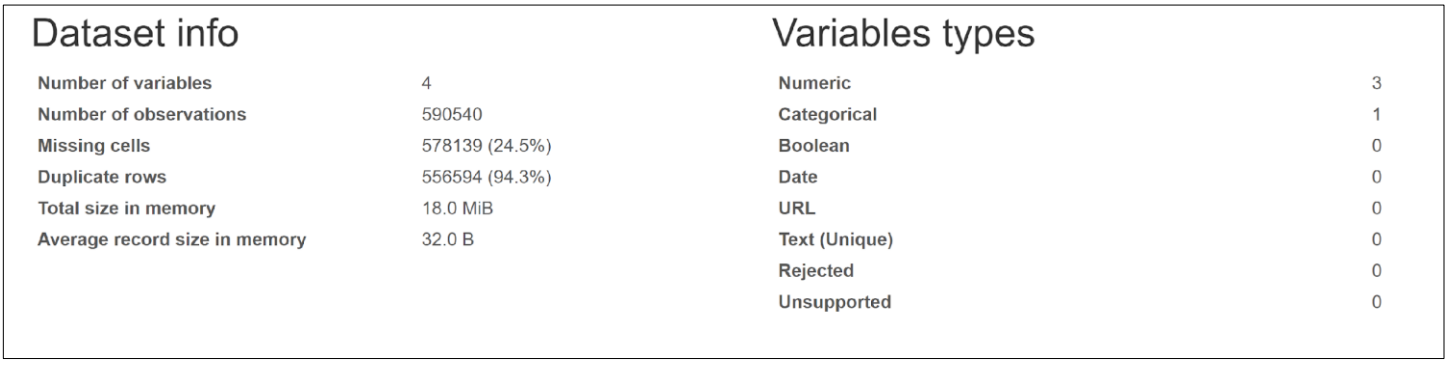
Data was already divided into train and test datasets. There were two separate files for transaction and identity data and these files are joined together based on transaction id.

Unbalanced Data:

Data was unbalanced with 96.5% of the transaction as non-fraudulent and 3.5% of the transactions as fraudulent, so we used parameter optimization and under sampling to maintain balance across training data. For our Logistic regression model, decision tree, and random forest we used under sampling to balance the data. For the light GBM model we applied baysian hyperparameter optimization to compensate for the missing data.

Pandas Profiling:

Pandas profiling is used during the project for elaborative exploratory data analysis. Below is details pandas profile report on Address variables.



Dataset info

Number of variables	4
Number of observations	590540
Missing cells	578139 (24.5%)
Duplicate rows	556594 (94.3%)
Total size in memory	18.0 MiB
Average record size in memory	32.0 B

Variables types

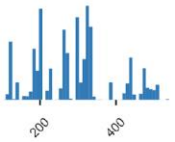
Numeric	3
Categorical	1
Boolean	0
Date	0
URL	0
Text (Unique)	0
Rejected	0
Unsupported	0

addr1 has 65706 (11.1%) missing values	Missing
addr2 has 65706 (11.1%) missing values	Missing
dist1 has 19824 (3.4%) zeros	Zeros
dist1 has 352271 (59.7%) missing values	Missing
P_emaildomain has a high cardinality: 60 distinct values	Warning
P_emaildomain has 94456 (16.0%) missing values	Missing

Variables

addr1
Numeric

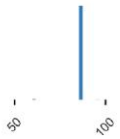
Distinct count	333	Mean	290.7337939
Unique (%)	0.1%	Minimum	100
Missing (%)	11.1%	Maximum	540
Missing (n)	65706	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		



Toggle details

addr2
Numeric

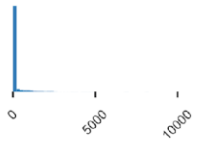
Distinct count	75	Mean	86.80063029
Unique (%)	< 0.1%	Minimum	10
Missing (%)	11.1%	Maximum	102
Missing (n)	65706	Zeros (%)	0.0%
Infinite (%)	0.0%		
Infinite (n)	0		



Toggle details

dist1
Numeric

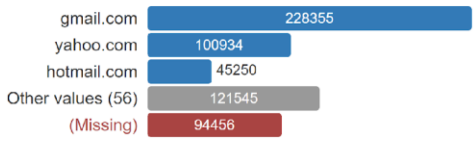
Distinct count	2652	Mean	118.5021803
Unique (%)	0.4%	Minimum	0
Missing (%)	59.7%	Maximum	10286
Missing (n)	352271	Zeros (%)	3.4%
Infinite (%)	0.0%		
Infinite (n)	0		



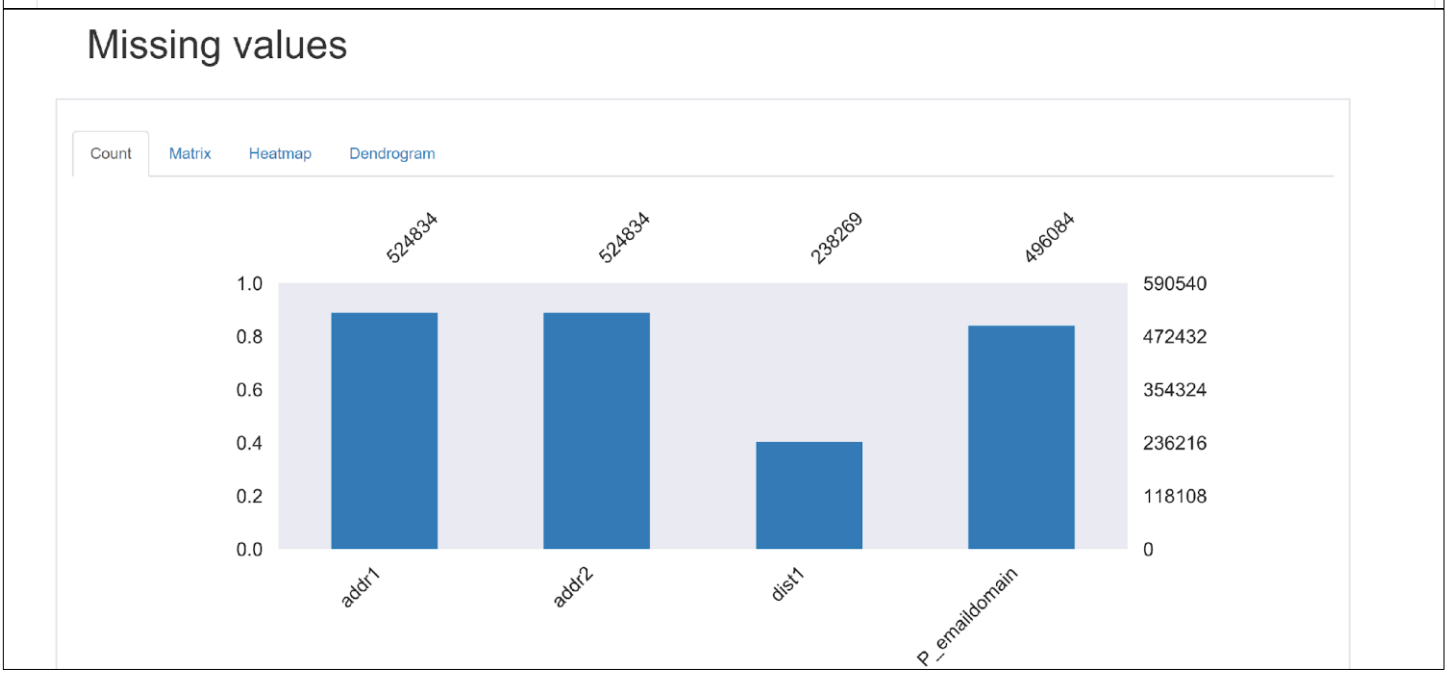
Toggle details

P_emaildomain
Categorical

Distinct count	60
Unique (%)	< 0.1%
Missing (%)	16.0%
Missing (n)	94456



Toggle details

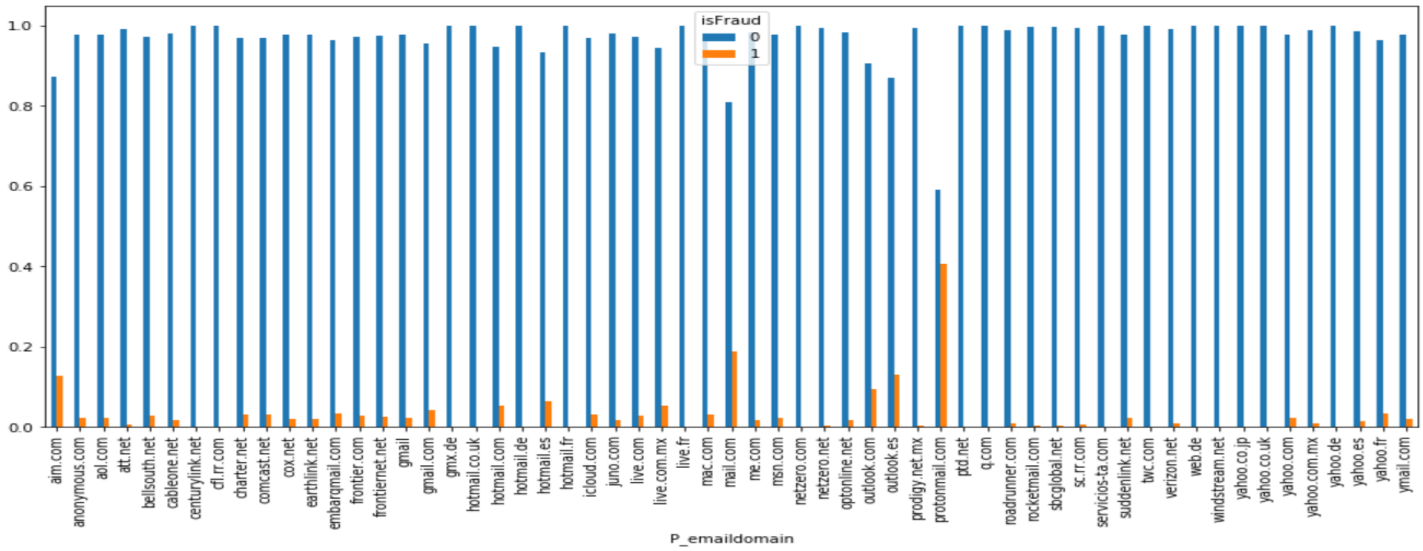


Inference:

It was observed that there were 60 distinct email domains and 16% of the samples had missing email domain and gmail was the most occuring email domain with 228355 occurences. Also, dist1 the variable with the more than 59% missing values.

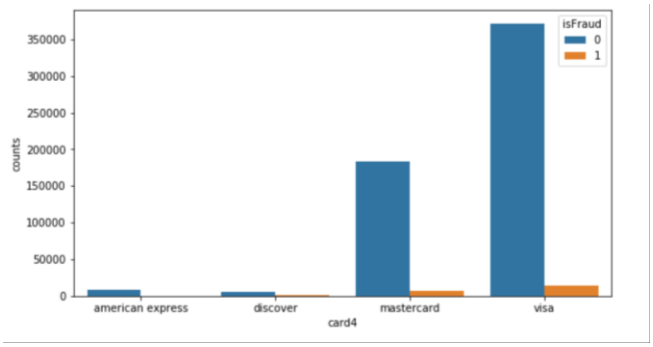
Fraudulent Email Domain:

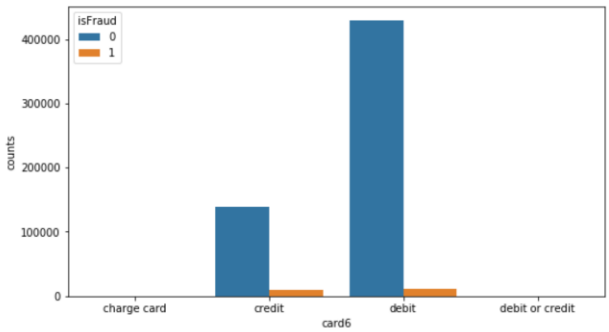
During data exploration it was observed that there were some fraudulent email domains in the data. Just list a few protonmail.com, mail.com.



Fraudulant transaction on type of card:

Mastercard and Visa card contributes to a higher percentage of fraudulent transactions





Feature Engineering

Feature engineering plays a very important role, especially when the dataset contains a large number of features. It helps prepare the proper input dataset, compatible with the machine learning algorithm requirements and hence improving the performance of machine learning models. Below are some of the feature engineering techniques that we used for this project:

- Created Aggregations on the features using mean and standard deviation to get more sense and logic from the data

```
[ ] train_df['TransactionAmt_to_mean_card1'] = train_df['TransactionAmt'] / train_df.groupby(['card1'])['TransactionAmt'].transform('mean')
    train_df['TransactionAmt_to_mean_card4'] = train_df['TransactionAmt'] / train_df.groupby(['card4'])['TransactionAmt'].transform('mean')
    train_df['TransactionAmt_to_std_card1'] = train_df['TransactionAmt'] / train_df.groupby(['card1'])['TransactionAmt'].transform('std')
    train_df['TransactionAmt_to_std_card4'] = train_df['TransactionAmt'] / train_df.groupby(['card4'])['TransactionAmt'].transform('std')
```

- Created new features by splitting the existing features(For example - Device name , Device version from Column Device Info or OS and OS Version from id_30 etc). This helped us gain more explicit insights from the data, for example which OS accounts for more fraudulent transactions etc

```
def id_split(dataframe):
    dataframe['device_name'] = dataframe['DeviceInfo'].str.split('/', expand=True)[0]
    dataframe['device_version'] = dataframe['DeviceInfo'].str.split('/', expand=True)[1]

    dataframe['OS_id_30'] = dataframe['id_30'].str.split(' ', expand=True)[0]
    dataframe['version_id_30'] = dataframe['id_30'].str.split(' ', expand=True)[1]
```

- Categorized the different models from the same company together, to avoid confusion. It helped make the analysis simpler as we did not observe any evident differences among different models of the same brand.

```
dataframe.loc[dataframe['device_name'].str.contains('SM', na=False), 'device_name'] = 'Samsung'
dataframe.loc[dataframe['device_name'].str.contains('SAMSUNG', na=False), 'device_name'] = 'Samsung'
dataframe.loc[dataframe['device_name'].str.contains('GT-', na=False), 'device_name'] = 'Samsung'
dataframe.loc[dataframe['device_name'].str.contains('Moto G', na=False), 'device_name'] = 'Motorola'
dataframe.loc[dataframe['device_name'].str.contains('Moto', na=False), 'device_name'] = 'Motorola'
dataframe.loc[dataframe['device_name'].str.contains('moto', na=False), 'device_name'] = 'Motorola'
```

- Used Count and Label encoding for categorical features in the datasets to make the data ready for modeling.

```
[ ]  for col in train_df.columns:
        if train_df[col].dtype == 'object':
            le = LabelEncoder()
            le.fit(list(train_df[col].astype(str).values) + list(test_df[col].astype(str).values))
            train_df[col] = le.transform(list(train_df[col].astype(str).values))
            test_df[col] = le.transform(list(test_df[col].astype(str).values))
```

- Dropped the correlated features using spearman and pearson coefficient values. Including correlated features in our model will give inaccurate model coefficients, accuracy and standard errors and therefore should not be included these features in the model.

- C1 is highly skewed (γ1 = 23.95795965) Skewed
- C10 is highly correlated with C1 (ρ = 0.9582021807) Rejected
- C11 is highly correlated with C10 (ρ = 0.9560557324) Rejected
- C12 is highly correlated with C11 (ρ = 0.9156926583) Rejected
- C13 has 35460 (6.0%) zeros Zeros
- C14 is highly correlated with C11 (ρ = 0.9624645831) Rejected

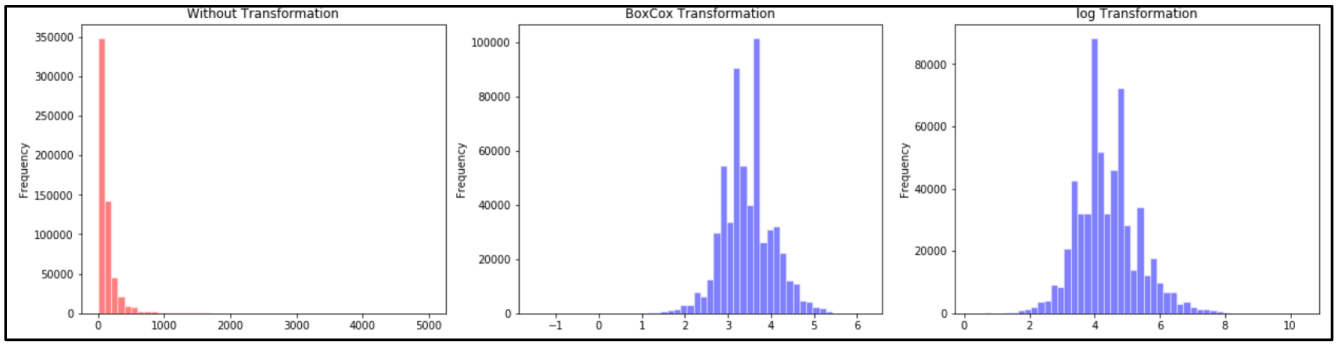
- Created new features such as day of the week and hour of the day to see the trend of fraudulent activities during different times. However, we did not see much difference in the fraudulent activity during the day of the week from the reference day or hour of the day.

```
[ ] train_df['Transaction_dow'] = np.floor((train_df['TransactionDT'] / (3600 * 24)-1) % 7)#day of a week from some referance day
    test_df['Transaction_dow'] = np.floor((test_df['TransactionDT'] / (3600 * 24)-1) % 7)#day of a week from some referance day
    test_df['Transaction_hour'] = np.floor(test_df['TransactionDT'] / 3600) % 24 # hour of the day
    train_df['Transaction_hour'] = np.floor(train_df['TransactionDT'] / 3600) % 24 # hour of the day
```

```
[ ] pd.crosstab(train_df['Transaction_dow'],train_df.isFraud).transpose()
```

Transaction_dow	0.0	1.0	2.0	3.0	4.0	5.0	6.0
isFraud							
0	94952	76871	67720	82746	82012	82410	83166
1	3550	2963	2503	2687	2803	2946	3211

- Created new transformed feature for TransactionAmt as it is highly skewed using BoxCox and log transformations to bring the data closer to the normal distribution.



Modeling:

We ran several models, Logistic Regressions, Decision Trees, Random Forest, and Light GBM + under sampling and Light GBM with parameter tuning. We chose not to run PCA analysis on our non-categorical variable to maintain visibility on which features were important.

LogisticRegression

```
[[117970  24444]
 [  1399   3822]]
```

DecisionTree

```
[[118321  24093]
 [  1865   3356]]
```

The best decision tree parameters are
{'max_depth': 3, 'max_leaf_nodes': 4, 'min_samples_split': 2}

RandomForestClassifier

```
[[115159  27255]
 [  1787   3434]]
```

LGBM_under sampling

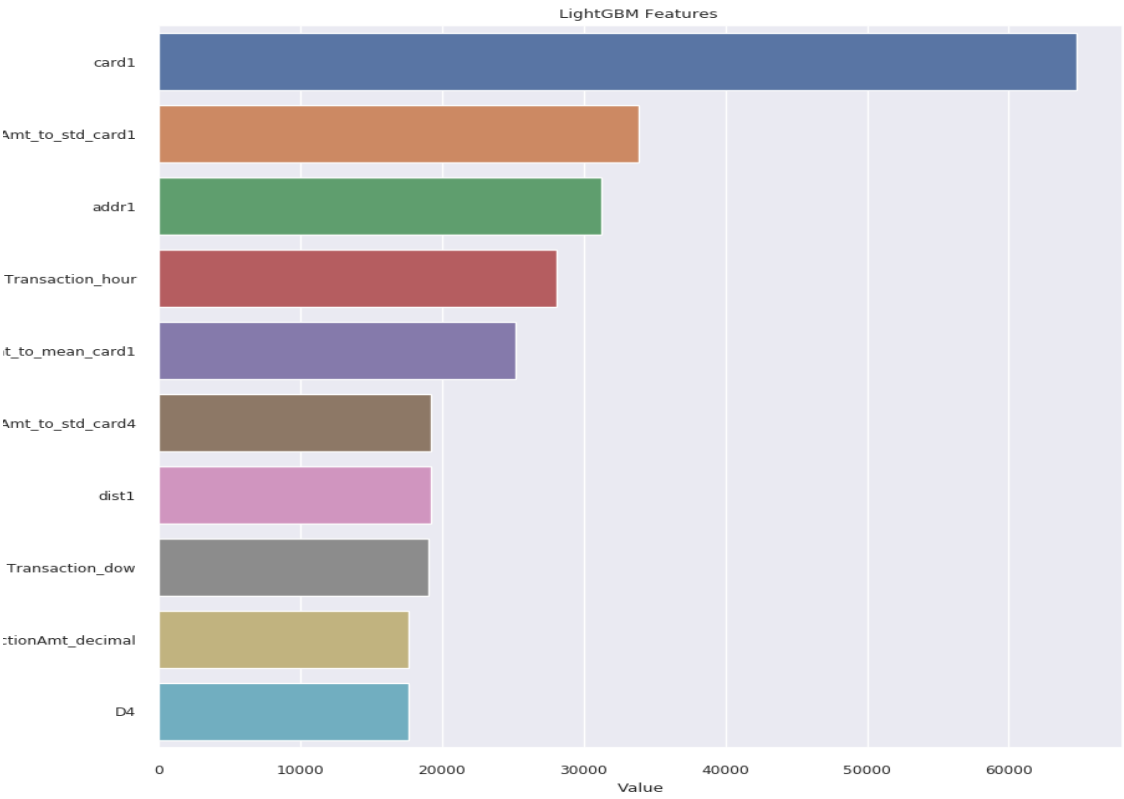
```
[[129567  12847]
 [   512   4709]]
```

LGBM with Bayesian hyperparameter optimization

```
[[142300    114]
 [  1510   3711]]
```

	Precision score	Recall score	Accuracy score	F1 score	Score
LogisticRegression	0.137	0.730	0.828	0.231	0.781
DecisionTree	0.144	0.518	0.874	0.225	0.703
RandomForestClassifier	0.111	0.657	0.803	0.191	0.733
LGBM_SMOOT	NaN	NaN	NaN	NaN	NaN
LGBM	0.970	0.711	0.989	0.820	0.976
LGBM_Undersampling	0.268	0.902	0.910	0.413	0.964

The model with the highest precision score is ['LGBM']
The model with the highest recall score is ['LGBM_Undersampling']
The model with the highest accuracy score is ['LGBM']
The model with the highest F1 score is ['LGBM']
The model with the highest AUC score is ['LGBM']

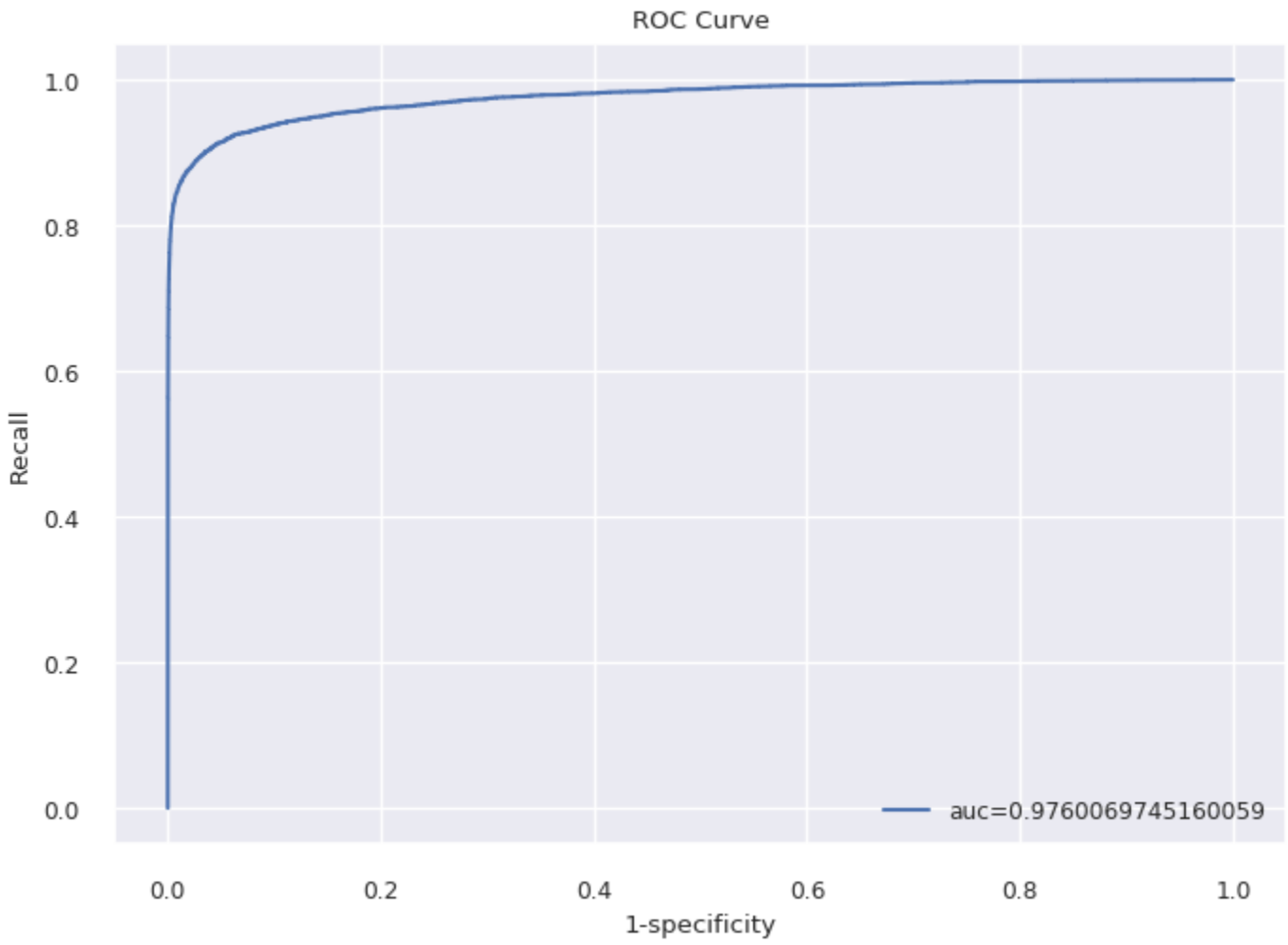


Inference:

The Logistic Regression, Decision Tree, and Random Forest required the data to be imputed and balanced. The NA and infinity values were imputed with 0 then under sampled. This same dataset was tested using Light GBM to validate the under-sampling methodology. Of the 4 models with under sampled data, Light GBM had the best model performance. The initial plan had SMOTE as part of the evaluation; however, this could not be accurate for our model because a large number of missing values were imputed to zero (SMOTE would create new data points based on our imputed zeros). Our under-sampled data had reasonably but had low precision across the board

Light GBM with parameter tuning, was able to compensate for the unbalanced data and the missing data. It did this by creating a spare matrix with NA instead of imputing them and ignoring them during the tree split then allocate them later

ROC curve for Light GBM



Conclusion

The best performing model was the LGBM with Bayesian hyperparameter optimization. This yielded both a high AUC value and Recall Score. The competition called for having the model with the highest score.

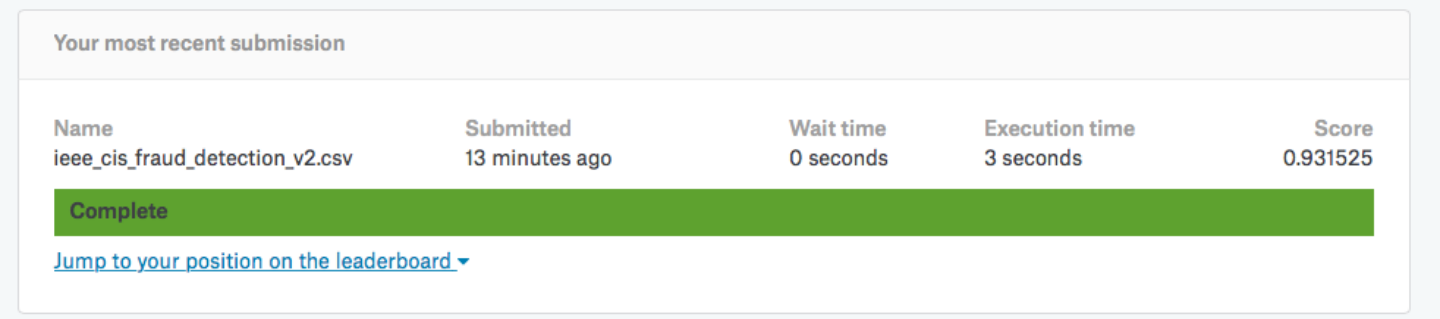
It's likely in the implementation of this model, the error scoring will need to take into account the cost of false positive and false negative. The cost of a false positive could cost the customer order, or a false negative which is the cost of the fraud going through.

While working on this project was a learning opportunity in itself. It helped us understand the importance of data and features on model accuracy and recall. Team performed various experiments with data and methodologies to produce the best solution. The model gave pretty good accuracy and recall score.

Kaggle Score

The submissions are evalutaed on area under the ROC curve between the predicted probability and the observed target.

Our final Score is listed below:



References

https://www.kaggle.com/c/ieee-fraud-detection/data