Whether a given credit card transaction will be fraudulent or not?

DSE I2100 Applied Machine Learning and Data Mining Final Project

Marjan Rezvani
Ayub ali Sarker
Maryam Akrami

Spring 2020

Abstract

One of the most important responsibilities that a bank or financial institution has is to protect the integrity of the institution by working hard to protect the financial assets that it holds. Bank fraud can be defined as an unethical and/or criminal act by an individual or organization to illegally attempt to possess or receive money from a bank or financial institution.
It is anticipated that card frauds would amount to around $30 billion worldwide by 2020. So, how banks can improve security by detecting and obstructing frauds? Machine learning is perfect for detecting frauds! Its algorithms learn to tell fraudulent transactions from legitimate operations.
In this project, we apply multiple ML techniques to the problem using card transaction data to identify fraudulent transactions (i.e. Fraud Detection)
We show that our proposed approaches are able to detect fraud transactions with high accuracy and reasonably low number of false positives.

Introduction

Banking Fraud has been an ever-growing issue with huge consequences to banks and customers, in terms of financial losses, trust and credibility.
An effective fraud detection system should be able to detect fraudulent transactions with high accuracy and efficiency.
A major challenge in applying ML to fraud detection is presence of highly imbalanced data sets. In many available datasets, majority of transactions are genuine with an extremely small percentage of fraudulent ones. Designing an accurate and efficient fraud detection system that is low on false positives but detects fraudulent activity effectively is a significant challenge.
In our project, we apply multiple binary classification approaches such as Logistic Regression, KNN, Random Forest and Voting Classifier to solve this problem on a labeled dataset that consists of payment transactions. our machine learning models collect information, analyzes the data gathered and extracts the required features. Our goal is to build binary classifiers which are able to separate fraud transactions from non-fraud transactions. We compare the effectiveness of these approaches in detecting fraud transactions.

Background

Several ML and non-ML based approaches have been applied to the problem of fraud detection which gave us some ideas to do our project.
For instance, The paper http://cs229.stanford.edu/proj2018/report/261.pdf reviews and compares such multiple state of the techniques, datasets and evaluation criteria applied to this problem. it applies multiple binary classification approaches - Logistic regression, Linear SVM and SVM with RBF kernel on a labeled dataset.
The paper https://ieeexplore.ieee.org/abstract/document/1297040 proposes a rule based technique applied to fraud detection problem.
The paper http://www.ecmlpkdd2018.org/wp-content/uploads/2018/09/567.pdf discusses the problem of imbalanced data that result in a very high number of false positives and proposes techniques to alleviate this problem.

Data

In this project we have used a dataset [https://github.com/msarker000/ml-group-project/blob/master/data/transactions.txt.zip] contains a year credit card transaction made in 2016. There are 641914 instances in the dataset. As it's shown below:

```
RangeIndex: 641914 entries, 0 to 641913
Data columns (total 29 columns):
accountNumber                641914 non-null object
accountOpenDate              641914 non-null object
acqCountry                   638001 non-null object
availableMoney               641914 non-null float64
cardCVV                      641914 non-null object
cardLast4Digits              641914 non-null object
cardPresent                  641914 non-null bool
creditLimit                  641914 non-null float64
currentBalance               641914 non-null float64
currentExpDate               641914 non-null object
customerId                   641914 non-null object
dateOfLastAddressChange      641914 non-null object
echoBuffer                   0 non-null float64
enteredCVV                   641914 non-null object
expirationDateKeyInMatch     641914 non-null bool
isFraud                      641914 non-null bool
merchantCategoryCode         641914 non-null object
merchantCity                 0 non-null float64
merchantCountryCode          641290 non-null object
merchantName                 641914 non-null object
merchantState                0 non-null float64
merchantZip                  0 non-null float64
posConditionCode             641627 non-null object
posEntryMode                 638569 non-null object
posOnPremises                0 non-null float64
recurringAuthInd             0 non-null float64
transactionAmount            641914 non-null float64
transactionDateTime          641914 non-null object
transactionType              641325 non-null object
dtypes: bool(3), float64(10), object(16)
```

This dataset is highly unbalance with a low percentage of fraudulent transactions within several records of non-fraud transactions, which is shown in figure 1. Out of the 641914 transactions in the dataset, 10892 were fraudulent which means the True frauds account for 1.7% of all transactions. The feature named 'isFraud' is used to classify the transaction whether it is a fraud or not, which takes value 1 in case of fraud and 0 otherwise. There are also features that describe customers' behavior are added, besides being fraud and not fraud. Some of numerical attributes are like available money, credit limit and categorical attributes like merchant name and transaction type. We also have few attributes such as echoBuffer, merchantCity, merchantState, merchantZip, posOnPremises,

recurringAuthInd, which totally have missing values that we dropped these columns. And for categorical features, we use Label Encoder to convert them to numeric.

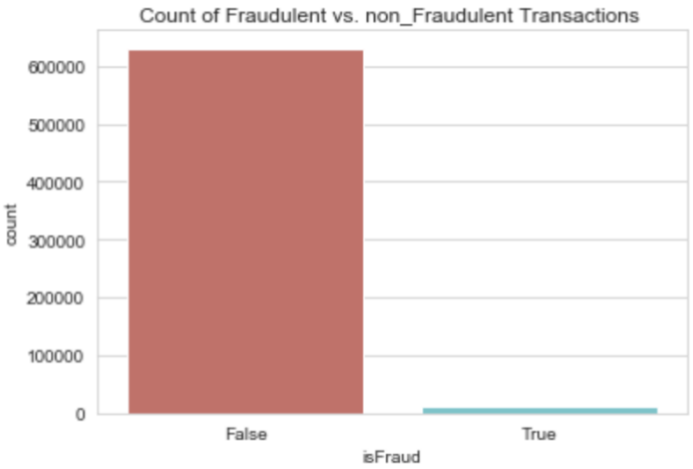


Figure 1

This graph(figure1) shows that the number of fraudulent transactions is much lower than the legitimate ones.

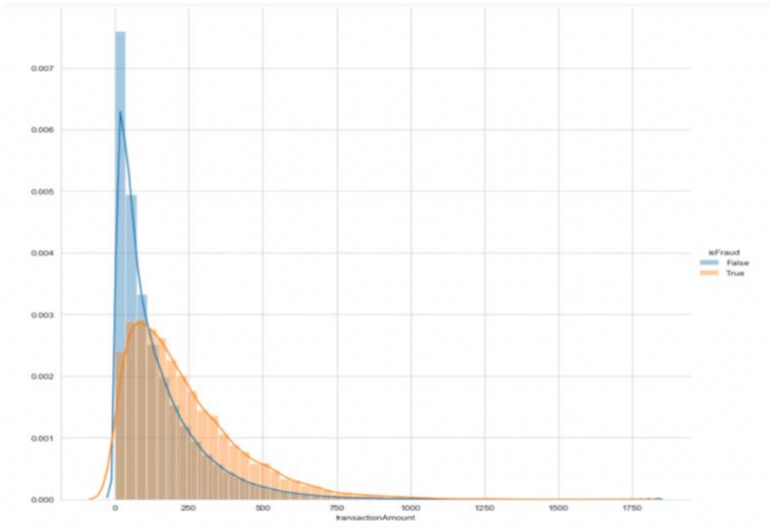


Figure 2

This graph(figure2) represents the amount that was transacted. A majority of transactions are relatively small and only a handful of them come close to the maximum transacted amount. And also, majority of the fraud transactions have transaction small amount.

Methods

Our goal is to detect fraud and non-fraud transactions by building appropriate binary classifiers. We have built classifiers using KNN, Random Forest and Voting Classifier. We compare the effectiveness of these approaches in detecting fraud transactions.
we have 29 features for 641914 transactions. Out of these transactions, only 10892 are labeled as fraudulent. With machine learning, our goal is to (1) learn a model that, given the features, can predict which transactions have this label, (2) evaluate the model and estimate its generalizable accuracy metric, and (3) identify the features most important for prediction. To achieve these three goals, we utilize a random forest classifier, which uses subsampling to learn multiple decision trees from the same data. We used scikit-learn's classifier with 500 trees by setting n estimators=500.
Each record has a number of attributes that describe it, and we can extract multiple features from this information alone. Some features are binary, like "cardPresent", "expirationDateKeyInMatch".
Majority of our features are categorical. These features are generated by converti ng categorical variables using LableEncoder. We also have few attributes which to tally have missing values, and as I mentioned them in data part, we dropped these features form the dataset.

- **Handle date and datetime features**

There are some datetime features we have in our dataset such as:

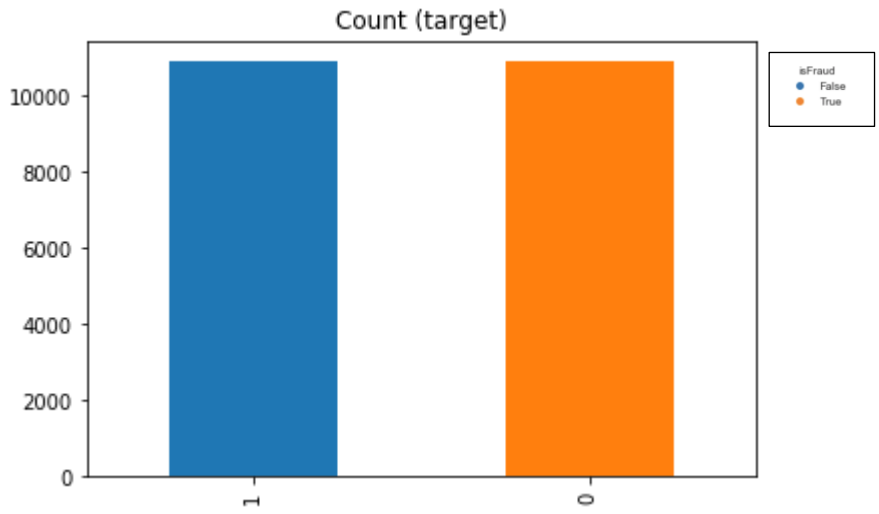
  - accountOpenDate
  - transactionDateTime
  - currentExpDate
  - dateOfLastAddressChange

We divide these features into several new features. This way is perfectly preserved intervals in easy intuition. For example, accountOpenDate is converted into accountOpenDate_year, accountOpenDate_month, accountOpenDate_day.

- **Resample to balance dataset**

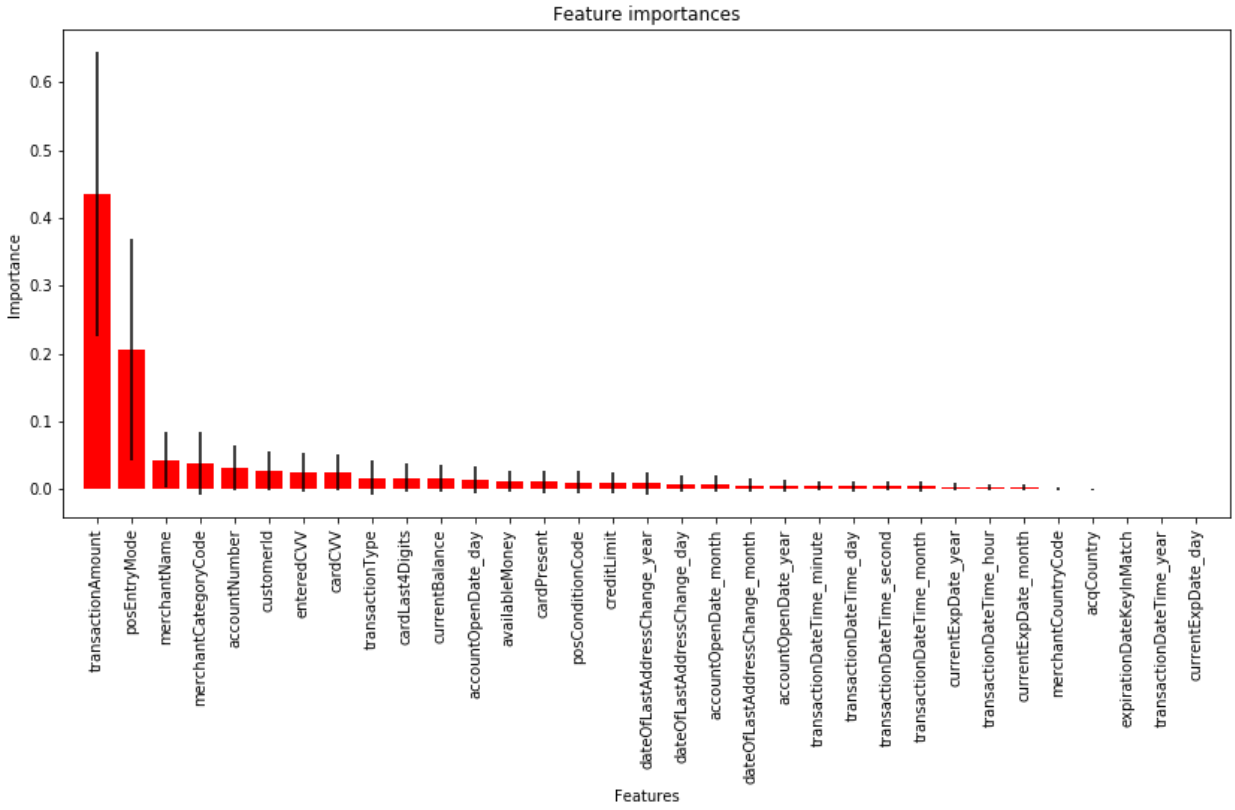
By looking into the figure 1, we can clearly see that our data is unbalanced. One of the major issues that users fall into when dealing with unbalanced datasets

relates to the metrics used to evaluate their model. Using simpler metrics like accuracy_score can be misleading. In a dataset with highly unbalanced classes, if the classifier always "predicts" the most common class without performing any analysis of the features, it will still have a high accuracy rate.
We did an experiment with this unbalanced dataset. We feed this dataset with all features into XGBClassifier and we got accuracy score 98.25%. We also feed again with only one feature and got accuracy score 98.25%. Which is a metric trap.
So, our dataset needs to be balanced before feed into any machine model.
Resampling consists of removing samples from the majority class(under-sampling) and / or adding more examples from the minority class (over-sampling).
In our case, we under sampled the large class (Not fraud) into small class (Fraud). After doing this we have now 50% large class and 50 % small class and our data is balanced (as it's shown in figure below).



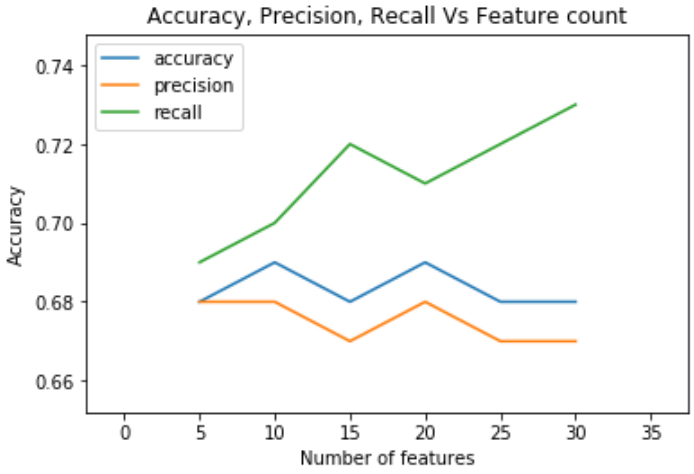
Feature Selection

Feature selection is important step in machine learning. By Feature selection we find important features those are most important in explaining the target variable. In our dataset we used RandomForestClassifier to see feature's importance. Here is the result we got by feed our dataset.



We see here transaction amount has higher importance, then posEntryMode, and merchantName and so on. Some feature's important are almost zero like acqCountry and merchantCountryCode.

To find top important sets of features we feed RandomForestClassifier to six different sets of top important features 5, 10, 15, 20, 25, 30. and plot the accuracy, precision and recall.



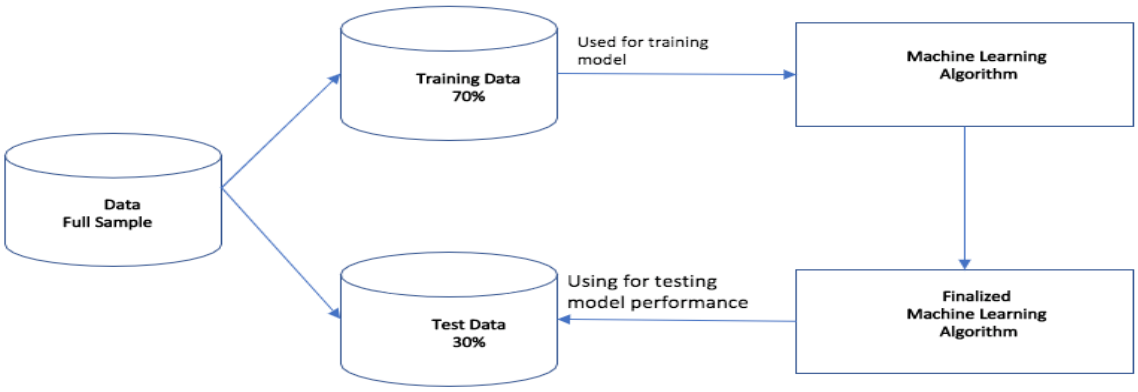
So, we see that accuracy, precision and recall for each 6 sets are almost same. but the set with 20 most important features has highest accuracy.
So, the followings are the high importance 20 features:

- transactionAmount (0.461284)
- posEntryMode (0.238895)
- merchantName (0.052225)
- merchantCategoryCode (0.042508)
- accountNumber (0.024675)
- customerId (0.022490)
- transactionType (0.021297)
- enteredCVV (0.018811)
- cardCVV (0.017843)
- cardPresent (0.016107)
- currentBalance (0.012183)
- accountOpenDate_day (0.011294)
- cardLast4Digits (0.010590)
- dateOfLastAddressChange_year (0.007816)
- dateOfLastAddressChange_day (0.007122)
- creditLimit (0.006090)
- accountOpenDate_month (0.005907)
- posConditionCode (0.005774)
- accountOpenDate_year (0.003972)
- availableMoney (0.003922)

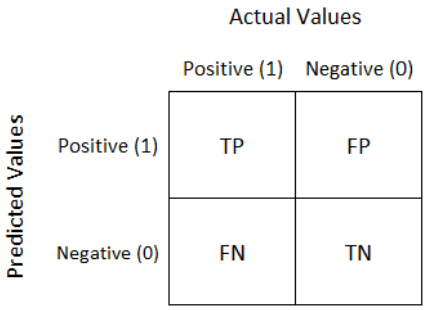
Evaluation

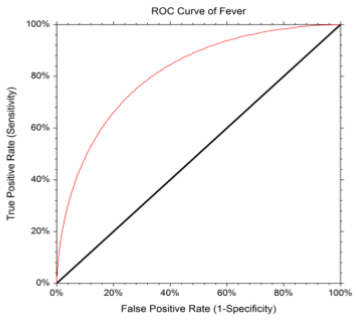
. Model training
. Hyperparameter tuning
. Evaluation

We investigated some supervised classification algorithms such as Logistic Regression, KNN, Decision Tree Classifier, Adaboost and Voting Classifier to solve this problem. To be able to test the performance of our algorithms, we standardized our preprocessed clean data and then split into train test split by the ration of 0.30 and feed into model.



Then we discuss results obtained in training, validation and testing phases. We evaluate performance of our models by computing metrics like recall, precision, f1 score, and area under curve (AUC).





we'll be using these abbreviations in the formulas:

$$Recall = \frac{TP}{TP + FN}$$

$$TPR, Recall, Sensitivity = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

$$FPR = 1 - Specificity = \frac{FP}{TN + FP}$$

- **Logistic Regression**

Logistic Regression is a classification algorithm which is helpful in calculating conditional probability.

Assume we have a supervised learning task where we are given M training instances $\{(X^i, y^i), i = 1, 2, \ldots, M\}$.

Here each $X^i \in \mathbb{R}^n$ is a n_dimensional feature vector, and $y^i \in \{0,1\}$ is a class label. Logistic regression models the probability distribution of the class label y given a feature vector X as follows:

$$\Pr(y = 1|X; \theta) = \sigma(\theta^T X) = \frac{1}{1+\exp(-\theta^T X)}$$

Here $\theta \in \mathbb{R}^n$ are the parameters of the logistic regression model and $\sigma(.)$ is a sigmoid function defined as above. Unknown parameters are estimated by maximizing likelihood function.

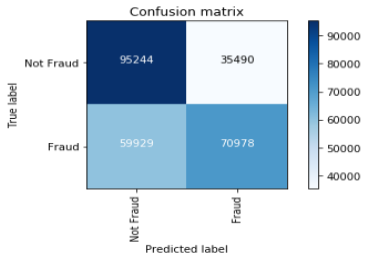
Parameter Tuning

After tuning hyperparameter for logistic regression using GridSearchCV, explore a range of parameters and finds the best combination of parameters as below:

{'C': 1, 'fit_intercept': True, 'penalty': 'l1', 'solver': 'liblinear'}

We fit our logistic regression model to our data and get the following classification report and confusion matrix when we tune its hyperparameters. The number of correct and incorrect predictions are summarized with count values and broken down by each class 'Fraud' and 'Not Fraud'.

	precision	recall	f1-score	support
Not Fraud	0.614	0.729	0.666	130734
Fraud	0.667	0.542	0.598	130907
micro avg	0.635	0.635	0.635	261641
macro avg	0.640	0.635	0.632	261641
weighted avg	0.640	0.635	0.632	261641



We observe 61.4% precision for 'Not Fraud' class and 66.7% for 'Fraud' class and average accuracy score 0f 63.5%
We have 95244+70978 correct predictions and 35490+59929 incorrect predictions.

- **KNN**

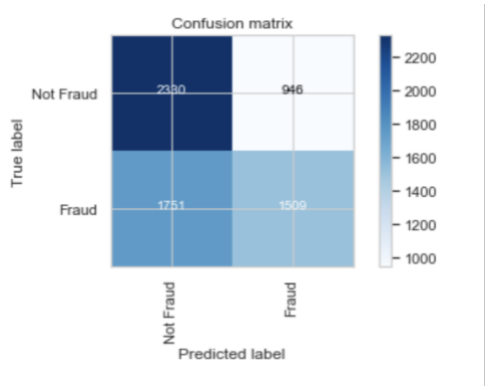
In KNeighborsClassifier we can tune number of neighbors, leaf size, metric (distance function/similarity metric) and n_jobs.

We did 10-fold cross validation and got accuracy score 58.9%
After running GridSearchCV, we got best score 59.6% and best parameters, {'leaf_size': 2, 'n_neighbors': 6}
Here is the classification report we got after feeding our dataset to best model

	precision	recall	f1-score	support
Not Fraud	0.571	0.711	0.633	3276
Fraud	0.615	0.463	0.528	3260
accuracy			0.587	6536
macro avg	0.593	0.587	0.581	6536
weighted avg	0.593	0.587	0.581	6536

We compared the result of Knn model with and without tuning hyperparameter, then found out that precision for 'Fraud' class has improved by 10% but there is still no significant improvement by best model. As we got accuracy score of 59.3%

Here is the confusion matrix we got by best model. As it's obvious in below figure, the confusion matrix demonstrates that TruePossitive is 2,330 and TrueNegative 1509 where FlasePositive is 946 and false negative is 1751.



- **DecisionTreeClassifier**

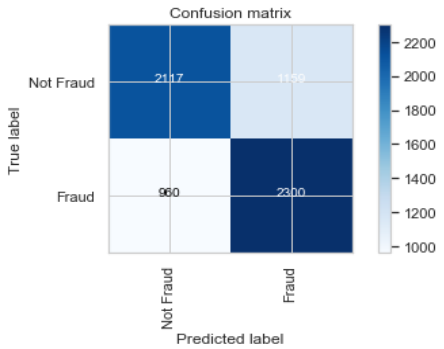
We feed the decision tree classifier(max_depth=5) without and with tuning. We also did 10-fold cross validation and got accuracy score 67.2%
In DecessionTreeClassifier we can tune:
-Decision tree in max depth (the depth of the tree)
-max feature (the feature used to classify)
-criterion
-splitter
GridSearchCV search explores a range of parameters and finds the best combination of parameters. Then repeat the process several times until the best parameters are discovered.
After running GridSearchCV, we got best score 67.1% and best parameters,

- 'criterion': 'entropy'
- 'max_depth': 6
- 'max_features': 20
- 'splitter': 'best'

Here is the classification report we got after feeding our dataset to best model:

	precision	recall	f1-score	support
Not Fraud	0.688	0.646	0.666	3276
Fraud	0.665	0.706	0.685	3260
accuracy			0.676	6536
macro avg	0.676	0.676	0.676	6536
weighted avg	0.676	0.676	0.676	6536

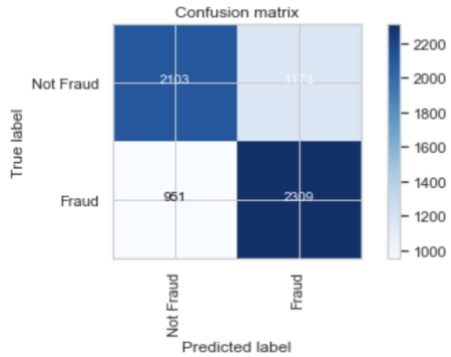
We found that there is no significant improvement by best model, as its accuracy score is 67.6%
And here is the confusion matrix we got by best model:



We see here we got TruePossitive is 2,117 and TrueNegative 2,300. And FlasePositive is 1,159 and false negative is 960.

- **Adaboost Classifier**

Adaboost initially selects a training subset randomly then iteratively trains the AdaBoost machine learning model by selecting the training set based on the accurate prediction of the last training.
We feed the Adaboost Classifier without and with tuning and got the 68.7% precision score for 'Not Fraud' class and 68% score for 'Fraud' class and average accuracy score is 68.4% without tuning.
Then after tuning such hyperparameters like base estimator, n_estimators, and learning_rate, running GridSearchCV, and got best score 68.8% and best params 'n_estimators': 50
And for the confusion matrix which is visualized as below:

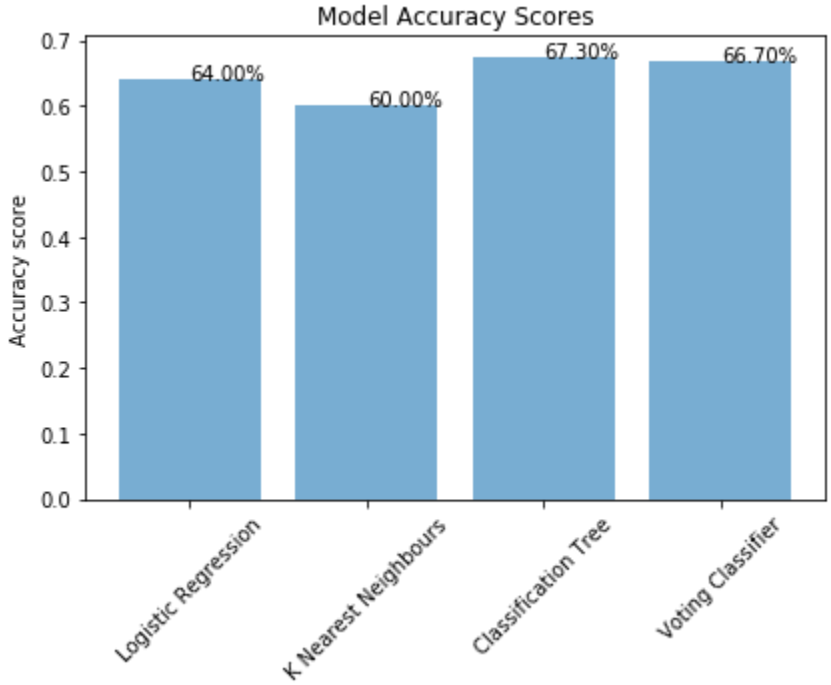


- **Voting Classifier**

The algorithm of voting classifier will allow us to combine different classifications into a single model, which is (ideally) stronger than any of the individual model alone. In our voting classing we combined three classifiers. LogisticRegression(64%), KNN(60%) and DecissionTreeClassifier(67.3%) and then feed our data to combine classifiers and got classification report as below:

	precision	recall	f1-score	support
Not Fraud	0.656	0.707	0.680	3276
Fraud	0.680	0.628	0.653	3260
accuracy			0.667	6536
macro avg	0.668	0.667	0.667	6536
weighted avg	0.668	0.667	0.667	6536

We see here 65.6% precision for “Not Fraud” class and 68% precision for “Fraud” class and accuracy score is 66.8%.
There is not too much improvement by voting classifier as we expected.



The figure above just shows the comparative accuracy for classifiers.
As we can see in the results, the ensemble classifier almost performs similarly on average.

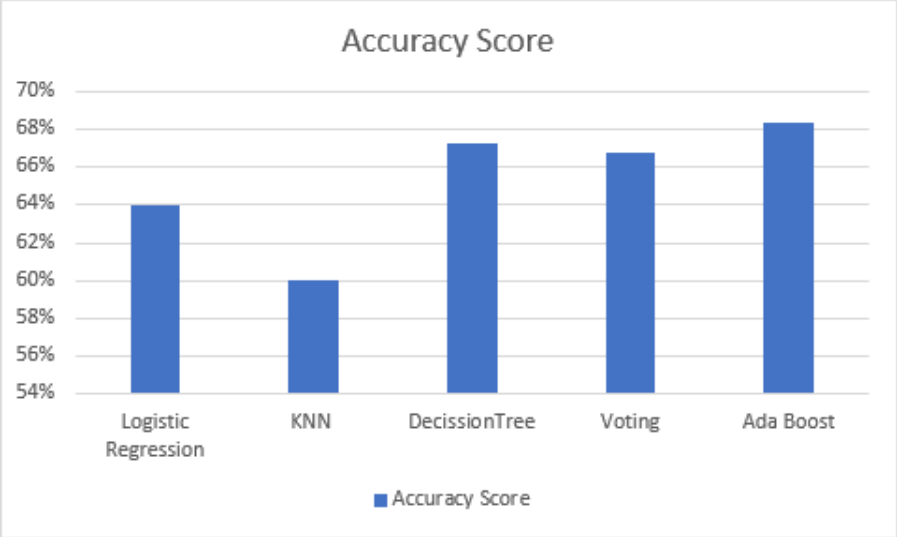
Result Comparison

In this section, we discuss results obtained in training, validation and testing sets. We evaluated performance of our models by computing metrics like recall, precision, and f1 score.

Fraud			
Algorithm	**Recall**	**Precision**	**F1_score**
Logistic Regression	**0.693**	**0.680**	**0.687**
KNN	**0.463**	**0.615**	**0.528**
Decision Tree	**0.706**	**0.665**	**0.685**
AdaBoost	**0.693**	**0.680**	**0.687**
Voting classifier	**0.628**	**0.680**	**0.653**

Not Fraud			
Algorithm	**Recall**	**Precision**	**F1_score**
Logistic Regression	**0.674**	**0.687**	**0.680**
KNN	**0.711**	**0.571**	**0.633**
Decision Tree	**0.646**	**0.688**	**0.666**
AdaBoost	**0.674**	**0.687**	**0.680**
Voting classifier	**0.707**	**0.656**	**0.689**

After training each of the models, these are the final results. All of the scores for Logistic Regression and the AdaBoost and Voting models are doing almost similarly for our dataset! Each model has a satisfied true positive rate and a low false positive rate, which is exactly what we're looking for.
Overall, we observe that all our proposed approaches seem to detect fraud transactions with an average accuracy.
As mentioned above, this project had the focus of achieving the highest accuracy. This is used to calculate the accuracy score of the algorithms. These results along with the classification report for each algorithm is given in a visualization of the comparative result that could easily be used to explain very simply why a decision was made about choosing a model:



We see that Decision Tree and Ada boost have higher accuracy compare to other classifiers and adaboost classifer has the highest accuracy score of 68%. Therefore, choosing Adaboost classifer might be a reasonable approach in order to achieve a higher degree of comprehensiveness while only slightly decreasing performance.

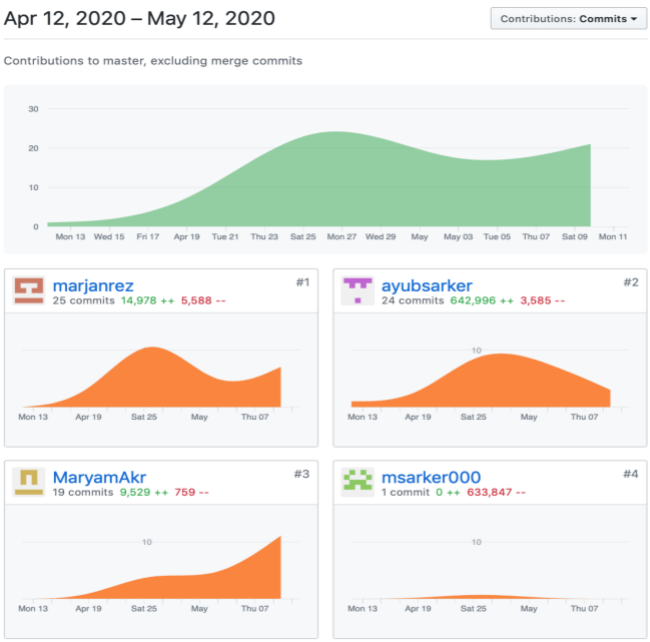
Conclusion

In fraud detection, we often deal with highly imbalanced datasets. For the chosen dataset, we showed that our proposed approaches are able to detect fraud transactions with reasonable accuracy and high precision which relates to low false positive rate. We found that the resulting Adaboost classifier was a little stronger and more robust than the individual scores.
As demonstrated before, the precision of the algorithms increases when the size of dataset is increased. Hence, more data will surely make the model more accurate in detecting frauds and reduce the number of false positives.
We can further improve our techniques by using algorithms like Decision trees to leverage categorical features associated with accounts/users in this dataset. We can create user specific models - which are based on user's previous transactional behavior - and use them to further improve our decision-making process. All of these, we believe, can be very effective in improving our classification quality on this dataset, and there is some room for improvement here.

Attribution

GitHub code link: https://github.com/msarker000/ml-group-project

Each member of our group contributed for every part of this project. It means everyone worked on all parts similarly, but at the end, one person finalized the works in one notebook. There are many varieties of commits as you can see in the GitHub for each person, but all of us worked similar amount of work overall.



Works of Members:

Marjan Rezvani:
I worked on different parts of our project consistently, which mainly includes parts below:

1. EDA
    - Data cleaning
    - Preprocessing data
    - Data wrangling
    - Label encoding
    - Feature engineering
2. Model
    - Logistic regression
    - Hyperparameter tuning
    - Resample data
    - Random forest
    - Decision tree
3. Evaluation models
4. Visualization
5. Main body of the documentations

Md Ayub Ali Sarker:

- Preprocessing (Partial)
- Feature Selection by Randon Forest Tree
- Dataset Balancing
- Decision Tree Classifier
    - Cross validation
    - Parameter tune by GridSearchCV
- Voting Classifier
- Managed Github

Maryam Akrami:
I have also spent plenty of time working on different parts of project, my main responsibilities are mentioned below:

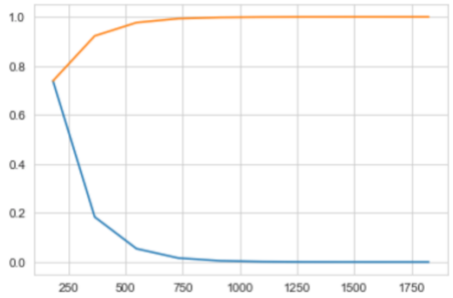
- Preprocessing data
    - focusing on handling the date time
- Implementing Models and fitting them to our data sets
    - Logistic Regression
    - KNN (K-nearest Neighbors)
    - Adaboost Classifier
    - Remodeling on balanced dataset
    - Hyperparameter tuning
- Model Evaluation
    - Evaluating Model’s performance based on accuracy scores
- Documentation
    - Involving in PowerPoints and reports

Bibliiography

[1] https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623
[2] https://github.com/JoyWang0320/Data-Mining_Credit-Card-Fraud-Detection/blob/master/CreditCard_YW.ipynb
[3] https://cs229.stanford.edu/proj2018/report/261.pdf
[4] https://scikitlearn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
[5] https://medium.com/@haydar_ai/learning-data-science-day-22-cross-validation-and-parameter-tuning-b14bcbc6b012
[6] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html

Appendex

In figure below, you can observe that probability of the points having transaction amount approximately less than 1750 is 1, it means almost all of the transactions have transaction amount less than 1750 and cdf curve verifies this fact.



We went through the instances and before converting the imbalance dataset to balance one, we noticed that fraud transaction has usually happened in which category.
Obviously, there is not any fraud transaction in ‘cable/phone’, ‘food_delivery’, ‘fuel’, ‘gym’, ‘mobileapp’ and ‘online_subscribtions’. And it happened mostly in ‘hotels’ transactions.