5291 Final Report
Fraud Detection Using Machine Learning Approaches

Xiaowei Chen - xc2597
Siying Liu - sl4980
Chun-Chieh,Tseng - ct3057
Zhihan Yang - zy2447
Yife Guo - yg2745

Department of Statistics
Columbia University in the City of New York

Spring 2022

Contents

1 Introduction

With the development of financial derivatives and business commercial products, the fraudulent rate of transactions has gotten increasingly higher in recent years, especially in e-commerce practices. Payment fraud is an illegal transaction defined by cybercriminals. That perpetrator deprives the funds or personal property of victims via the internet. There are three normal ways of payment fraud, unauthorized transactions, lost merchandise, and false requests for refund or return. Among all three-way to fraud, we identify the pattern of each type of transaction that could be listed as a reversed transaction, multi-swipe, and unauthorized transaction. Since the dataset we have listed all transactions from one source records, it's hard to know if the transaction is authorized without access to the banking's internal system. All in all, it's quite crucial to set up an accurate and efficient detecting system to identify potentially fraudulent transactions for banks, that have thousands of transactions happening each day, to better monitor irregular transaction activities.

Without touching the confidential client information, our project aims to develop a machine learning model to detect fraudulent transactions with specific features, like reversed transactions and multi-swipe transactions. We compare that feature with legitimate transactions to detect fraudulent items while having expected precision and accuracy from each machine learning algorithm we implement for this subject.

Our goal of this project is to present the significance of the fraud detection machine learning model that helps with detecting patterns of transaction or customer behaviors in financial operations. This model would significantly help banks and financial institutions to decide whether a given transaction is legitimate or fraudulent. So, institutions that leverage our machine learning model could be beneficial from fraud detection while running their business. More importantly, fraud detection machine learning models are more effective than humans, and they could handle overloaded data in a short period of time. They could somehow beat the traditional fraud detection system, like human labeling.

In this project, we trained five models including supervised and unsupervised learning models, like Logistic Regression, Random Forest, Neural Network, Lightgbm, and AdaBoost, while we also attached model evaluation in the table at the end to assess the performance of each model.

2 Data Description

The data contains 786363 transaction records with the dependent variable isFraud marked whether each transaction is Fraud or not. In the dataset, there are 28 variables related with 4 continuous variables with data type float, 22 categorical variables with data type object, and two boolean variables: The resource of data is retrieved from One Capital and downloaded from Kaggle. Dictionary for the dataset:

Variable Name	**Meaning**
accountNumber	Individual account number
customerId	Individual ID
creditLimit	Individual Credit Limit
availableMoney	Individual available Money Left
transactionDateTime	Transaction Date and Time
transactionAmount	Transaction Amount
merchantName	Merchant Name
acqCountry	Merchant Origin Country
merchantCountryCode	Merchant Country using Code
posEntryMode	How is the purchase being made
posConditionCode	Whether pos working normally
merchantCategoryCode	Which kind of Merchant it is
currentExpDate	Credit Card Expiring Date
accountOpenDate	Credit Card Open Date
dateOfLastAddressChange	Last date the card holder change its address
cardCVV	Credit Card CVV
enteredCVV	Binary Variables whether CVV is used
cardLast4Digits	Last 4 digits of the Credit Card
transactionType	Transaction Types
echoBuffer	Response Time after Purchasing
currentBalance	Current Credit Card Balance
merchantCity	City where the Merchant locates
merchantState	States where the Merchant Locates
merchantZip	Zip Code where Merchant Locates
cardPresent	Whether Physical Credit Card is Used
posOnPremises	
recurringAuthInd	
expirationDateKeyInMatch	Expiration

Table 1: Dictionary for the dataset

2.1 Dependent Variable:

Taking a glance of the target variable transaction that we want to take attention of is the Fraud transaction, and it consists of 1.58% of the overall transactions (Figure 1), which creates an imbalance dataset, and based on the density plot of Processed Amounts for Fraud and Not Fraud(Figure 1), most of the transactions in our dataset are small amount, and transactions flagged as not fraud shows right-skewed distribution. And transactions that are not fraud do not have the kurtosis, and most transactions consider having less amount of money.

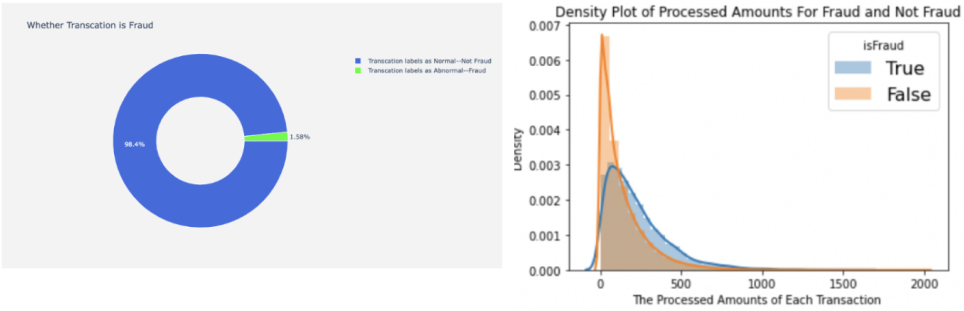


Figure 1: Proportion and density plot for fraud and not fraud

2.2 Independent Variable:

In the dataset, there are 4 money-related variables that may draw attention. Those are credit limit, available Money, transaction amount, and current balance. There is a strong correlation between the credit limit and available money, as well as credit limit and current balance, suggesting that further action is required(Figure 2). The following table shows basic statistics of the Money-Related variables (Figure2).

Figure 2: Conrrelation matrixs among variables

	creditLimit	availableMoney	transactionAmount	currentBalance
count	786363	786363	786363	786363
mean	10759.46	6250.73	136.99	4508.74
std	11636.18	8880.79	147.73	6457.44
min	250.00	-1005.63	0.00	0.00
25%	5000.00	1077.42	33.65	689.91
50%	7500.00	3184.86	87.90	2451.76
75%	15000.00	7500.00	191.48	5291.10
max	50000.00	50000.00	2011.54	47498.81

Table 2: Numerical variables description

2.3 Transaction by Time of the Day or Year

By further investigating whether a specific time of the day or specific time of the month would have a higher chance of fraud transaction, then all fraud transactions dataset was selected and plotted in the following histogram. It's not clear to see if a specific time of day would generate fraudulent transactions, along with the histogram of xxx, the month of the year does not show an obvious trend in processing fraudulent transactions.

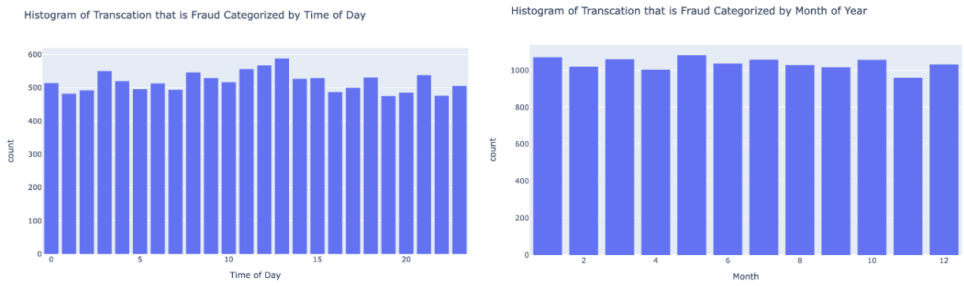


Figure 3: Number of transactions distribution in a day/ year

3 Data Wrangling - Duplicate Transactions

3.1 Reversed transactions

Based on the data set there is one thing to notice is that TranscationType consists of 4 different categories: None labeled, Address Verification, Purchase, and Reversal. While taking a deep look at the reversal transactions, there are 20,303 transactions that can be regarded as reversed transactions, which consists of 2.6% of the total transactions. And from those transactions, gap.com, amazon.com, staples.com are three merchants that are more likely to process reversed transactions.

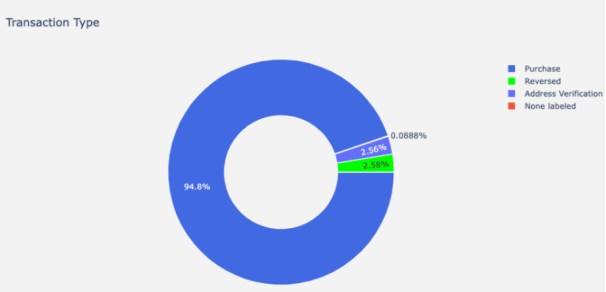


Figure 4: Reversed transaction percentage

3.2 Multi-swipe transactions

Meanwhile, we notice that some transactions were made with the same amount at the same merchant, meaning there existed possible multi-swipe transactions. Then we assume that if more than one transaction with the same amount was processed within one day by the same amount, or the time difference between the two transactions is less than 6 minutes, then we determine the transaction as a multi-swipe transaction. Among the dataset, there are 12,973 transactions that can be regarded as multi-swipe transactions, where gap.com, sears.com, and staples are the three transactions that are most easily processed by multi-swipe transactions.

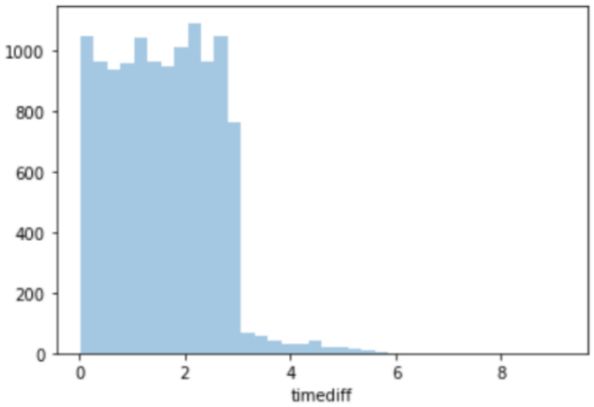


Figure 5: Multi- swipe transactions

4 Model

4.1 Data Preprocessing

Inside the dataset, 11 variables have missing values and 6 empty variables whose number of uniques is 1. According to Figure 3.1, the amount of NA values is lower than one percent which is a relatively small amount compared with the total data points. Thus, deletion is used to solve the problem. In addition, there is no duplicated row.

4.2 Feature Engineering

Looking at the isFraud value more detailedly, we can see that fraud transactions occurred more 10-15 minutes after the transaction compared with normal transactions according to Figure 3.2. In addition, from the correlation plot between independent variables grouped by dependent variable isFraud (Figure 3.3), availableMoney and currentBalance is negatively correlated. Through the insight of isFraud, transaction amounts correlated with other variables are all right-skewed and fraud transactions are more likely to happen when the transaction amount is small.

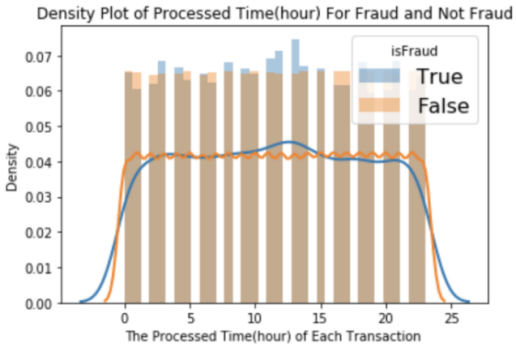


Figure 6: Density Plot of Processed Time(hour)

4.3 New Features Generation

After having a thorough understanding of the dataset, we create 7 new variables that would help us to better analyze important features associated with fraud transactions. They are CVV_ isright, hour, weekday, years_till_card_expire, years_till_address_change, transaction_history_N, transaction_history_mean.

- CVV_isright is a boolean variable that verifies whether CVV on the card is the same as the one entered. This will help us to check if the fraud transaction is caused due to wrong CVV entered.

- Hour is generated by the transaction date. We will be able to know if fraud transactions happened more frequently in how many hours after the transaction.

- Weekday is generated by the transaction date as well. This variable is important since we can observe the pattern of transactions. For example, fraud transactions are likely to happen during weekends compared to weekdays since people have more time to go shopping during the weekends.

- Years_till_card_expire calculated the time difference between the date of transactions and the date of card expiration. This will help to see if a fraud transaction is correlated with the card

expiration date. For example, people have a higher possibility of leakage of information since the card has been exposed more.

- Years_till_address_change is examined by the time difference between the date of transactions and the date of last address change. This concern is similar to Years_till_card_expire. Because changing addresses may provide some chances for other people to obtain your card information resulting in fraud transactions.

- Transaction_history_N counts the number of transactions by an account number. This variable allows us to examine whether fraud transactions have a higher possibility to happen when people swipe more cards.

- Transaction_history_mean calculated the average amount of transactions by the same account. Similar to Transaction_history_N, we will be able to know if fraud transactions are more likely to happen when the transaction amount is large or small.

After adding new variables, currently, we have 77668 data points with 30 variables. Then, we transfer all the categorical variables into dummy variables. For boolean variables, we assume 0 as True and 1 as False in order to make the modeling process more convenient.

Finally, we check the correlation matrix again to see if there exists multicollinearity. Multicollinearity occurs when there are near-linear dependencies among regressors. It is undesirable as the estimation of one regressor's impact on the dependent variable tends to be less precise than if predictors were uncorrelated with one another. According to Figure 3.4, on average the correlation is smaller than 0.05, which indicates that there is no obvious relationship between independent variables and multicollinearity does not exist.

4.4 Oversampling and Undersampling

Since data are extremely imbalanced, with 764, 702 class-0 records and 11,966 class-1 records, the minority class only takes a portion of 1.54% of the whole dataset. Therefore, our model might ignore class 1, which classifies all test data into class 0 and reaches an accuracy of around 98.46%, but 0 precision and 0 recall.

That's very ideal, because, in real life, the situation of fraud seldom happens, but ignoring fraud would result in severe problems. So, we need to do some tricks on our dataset to make it fairly balanced and make our prediction more stable.

Generally, there are 2 major ways to deal with imbalance situations, one is to give weights to each class and the other is random resampling. And the shortcoming of the first approach is that finding a proper weight is time-consuming, especially in this extreme scene. What's worse, giving a reasonable explanation for this manual weight is more difficult than finding it.
Then we introduce a second method. Resampling involves creating a new transformed version of the training dataset in which the selected examples have a different class distribution. And the simplest strategy is choosing examples in the dataset randomly, hence we call it random resampling.

There are 2 main ways to random resampling for imbalanced classification data, they are oversampling and undersampling.

- Random oversampling: randomly duplicate examples in the minority class

- Random undersampling: randomly delete examples in the majority class

2 ways are both feasible, but taking our dataset into consideration, if we apply random undersampling, we will discard around 752,736 records in total, which take 98.44% of all major classes and the size of our data shrinks sharply from 776,668 to 23,932. This means that we drop a huge amount of useful information. Since our data has 50 attributes and we will build some complex models like Random Forest, LightGBM, neural networks, etc., this amount of data records is far more than enough to fit those models. Hence, we use random oversampling to solve the issue of imbalance.

Just duplicating examples from the minority class in the training dataset is a fast way of oversampling, but this provides no more information than the original data. A technique called Synthetic Minority Oversampling Technique, or SMOTE for short will improve the duplicated dataset.

SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space, and drawing a new sample at a point along that line.

It first selects a minority class instance at random and finds its k nearest minority class neighbors. The synthetic instance is then created by choosing one of the k nearest neighbors b at random and connecting a and b to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances a and b.
The approach is effective because new synthetic examples from the minority class are created that are plausible, that is, are relatively close in feature space to existing examples from the minority class.

A general downside of the approach is that synthetic examples are created without considering the majority class, possibly resulting in ambiguous examples if there is a strong overlap between the classes.

Finally, we randomly split the dataset into two parts with 80 percent of data as a training set and 20 percent of value as the test set. For the training data, we used the SMOTE algorithm to balance the proportion of 0 and 1. Since the test set is used to test whether our final model is a good fit, we do not need to adjust.

5 Model training

5.1 Logistics Regression

Model description
Logistic regression is a useful analysis method for classification problems, where you are trying to determine if a new sample fits best into a category. It can predict the probability of a discrete outcome given an input variable. The most common dependent variable in logistic regression is binary, such as true/false, yes/no. Multinomial logistic regression model scenarios where there are more than two possible discrete outcomes, which will be a good fit for our data.

Model fitting
First, we build a simple logistic model without fine-tuning. From the previous oversampling step, we finally have a balanced dataset with both 611749 values of 0 and 1, therefore, we take the default threshold = 0.5. After cross-validation, the accuracy of the logistic regression classifier is 0.71, which is relatively lower than we expected.
In order to improve the accuracy and make a better fit, we use regularization, a technique used to prevent overfitting. It adds a penalty to the model in order to prevent the overfitting of the model. There are two common penalties: L1 regularization is called Lasso Regression and L2 is known as Ridge Regression.

- Lasso is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces.
- Ridge regression shrinks the coefficients and it helps to reduce the model complexity and multi-collinearity.

For the fine-tuning, there are several parameters we can adjust:

- Tol: Tolerance
- C: Inverse of regularization strength, smaller values specify stronger regularization.

After fine-tuning the penalty [l1, l2], C [1, 0.1, 0.01], tol [0.01,0.1,0.5] with cross-validation 5 fold, we obtain the final model LogisticRegression(C=0.1, penalty= 'l2', tol=0.5, solver="saga").

However, the accuracy of the final model is 0.55 (Figure 4.1.2) which is lower than the simple model before fine-tuning. According to the evaluation matrix and confusion matrix (Figure 4.1.1 - 4.1.4), the ACC and AUC value of the model before fine-tuning is higher than the one after fine-tuning. So we finally chose the first model.

One possible reason is that the dataset does not have many dependent variables which will not be complicated enough to use LASSO. In addition, the data does not have multicollinearity, so Ridge may not have a significant effect on the model. Another limitation is that the data is extremely unbalanced. Although we use the SMOTE algorithm to oversample, the amount of 1 value is duplicated which still may result in relatively inaccurate test results.

5.2 Random Forest

Model description
Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees.

For regression tasks, the mean or average prediction of the individual trees is returned.

Advantages:

- It reduces overfitting in decision trees and helps to improve the accuracy
- It is flexible to both classification and regression problems
- Can be used to automate missing values present in the data
- Normalizing of data is not required as it uses a rule-based approach
- Can automatically balance data sets when a class is more infrequent than other classes in the data
- Train a model with a relatively small number of samples and get pretty good results
- Random Forest is comparatively less impacted by noise

Disadvantages:

- Hard to interpret
- Fails when there are rare outcomes or rare predictors, as the algorithm is based on bootstrap sampling
- A large number of trees can make the algorithm too slow and ineffective for real-time predictions.

Hyperparameters:

- n_estimators: The number of trees in the forest
- criterion: measure the quality of a split. "gini impurity" or "information gain"
- max_depth: to control the size of the tree to prevent overfitting
- min_samples_split: The minimum number of samples required to split an internal node
- max_features: The number of features to consider when looking for the best split. (Sqrt(n_features), log(n_features), n_features)
- class_weight: The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y)). For multi-output, the weights of each column of y will be multiplied. Note that these weights will be multiplied with sample_weight (passed through the fit method) if sample_weight is specified. (balanced)
- n_jobs: The number of jobs to run in parallel
- max_samples: If bootstrap is True, the number of samples to draw from X to train each base estimator

Model fitting
We use 3 folds CV and grid search to fit the random forest model, after fine-tuning the model, we find the best hyper-parameters should be

- max_depth: 3
- max_features: 3
- min_samples_splith: 2
- n_estimators: 50

Then, we output the feature importance of fitting random forest models. We can tell from figure 7 that the transaction amount is the most important feature to determine whether it is a fraud transaction. Besides, May and September are two months that are important to consider.

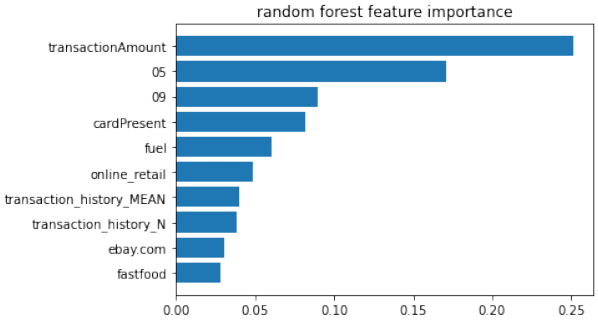


Figure 7: Random Forest feature importance

5.3 Lightgbm

Model Description
Light GBM 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. In other words, Light GBM grows trees vertically while other algorithms grow trees horizontally.

Advantages:

- Faster training speed and higher efficiency: Light GBM uses a histogram-based algorithm i.e it buckets continuous feature values into discrete bins which fastens the training procedure.
- Lower memory usage: Replaces continuous values with discrete bins which results in lower memory usage
- Better accuracy than any other boosting algorithm: It produces much more complex trees by following a leaf-wise split approach rather than a level-wise approach which is the main factor in achieving higher accuracy
- Compatibility with Large Datasets: It is capable of performing equally well with large datasets with a significant reduction in training time as compared to XGBoost.
- Parallel learning is supported
- Can deal with categorical data by pointing them out

Disadvantages:

- Overfitting: Light GBM splits the tree leaf-wise which can lead to overfitting as it produces many complex trees.
- Compatibility with Datasets: Light GBM is sensitive to overfitting and thus can easily overfit small data.

Hyperparameters:

- *application: default=regression, type=enum, options= options*
- *regression : perform regression task*
- *binary : Binary classification*
- *multiclass: Multiclass Classification*
- *lambdarank : lambdarank application*
- *data: type=string; training data , LightGBM will train from this data*
- *num_iterations: number of boosting iterations to be performed ; default=100; type=int*
- *num_leaves : number of leaves in one tree ; default = 31 ; type =intdevice : default= cpu ; options = gpu,cpu. Device on which we want to train our model. Choose GPU for faster training*

- *max_depth: Specify the max depth to which the tree will grow. This parameter is used to deal with overfitting*
- *min_data_in_leaf: Min number of data in one leaf*
- *feature_fraction: default=1 ; specifies the fraction of features to be taken for each iteration*
- *bagging_fraction: default=1 ; specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting*
- *min_gain_to_split: default=.1 ; min gain to perform splitting*
- *max_bin : max number of bins to bucket the feature values*
- *min_data_in_bin : min number of data in one bin*
- *num_threads: default=OpenMP_default, type=int ;Number of threads for Light GBM*
- *label : type=string ; specify the label column*
- *categorical_feature : type=string ; specify the categorical features we want to use for training our model*
- *num_class: default=1 ; type=int ; used only for multi-class classification*

Model fitting
After resampling our data, we used grid search and a 3-fold cross-validation method to fine-tune the hyperparameters. We chose 'precision' as the evaluation metric in the cross-validation method. The hyperparameters that we tune are :

- *'num_leaves': sp_randint(6, 50),*
- *'min_child_samples': sp_randint(100, 500),*
- *'min_child_weight': [1e-5, 1e-2, 1, 1e1, 1e3, 1e4],*
- *'subsample': sp_uniform(loc=0.2, scale=0.8),*
- *'colsample_bytree': sp_uniform(loc=0.4, scale=0.6),*
- *'reg_alpha': [0, 1e-1, 1, 2, 5, 7, 10, 50, 100],*
- *'reg_lambda': [0, 1e-1, 5, 10, 20, 50, 100],*
- *'boosting_type' : ['gbdt'],*
- *'learning_rate': [0.05],*
- *'colsample_bytree' : [0.5,0.7],*
- *Subsample':[0.1]*

Which regularized the size and depth of the trees and added regularization terms into the model. The final parameters that was selected by the cv method are:

- *Colsample_bytree = 0.7,*
- *Learning_rate = 0.05,*
- *Min_child_samples = 301,*
- *Min_child_weight = 10.0,*
- *N_estimators = 500,*
- *Num_leaves = 48,*
- *Subsample = 0.1*

We used the best parameter to retrain the training data, and finally achieved an accuracy of 0.95. Then, we output the feature importance of fitting lightGBM models. We can tell from figure 8 that the transaction history is the most important feature to determine whether it is a fraud transaction. Besides, current balance and available money are two important features to consider.

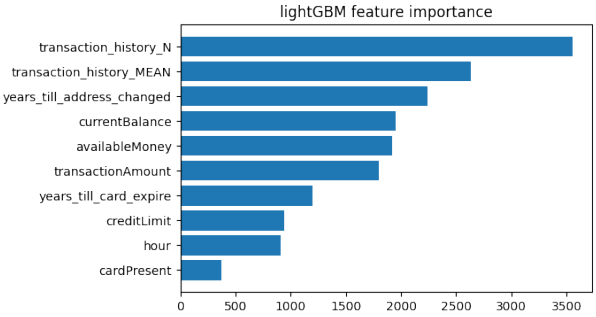


Figure 8: LightGBM feature importance

5.4 Adaboost

Model description
Adaptive Boosting is a Machine Learning technique that is used here as part of an Ensemble Method. We utilize the Adaboost Classifier since Adaboost is commonly used for classification tasks. We use Adaboost as one of the machine learning models to categorize the fraudulent transaction, as well as to validate the correctness of other models' performance.

The model is built on the basis of randomness. This means that the method will yield somewhat different results each time it is applied to the data. It is best practice to evaluate machine learning algorithms with a stochastic learning algorithm by averaging their performance across numerous runs or cross-validation repeats. When fitting a final model, it may be preferable to either increase the number of trees until the model's variance is lowered over time or to fit many final models and average their predictions.

Model fitting
n estimators: The number of weak learners or base estimators we'd like to utilize in our dataset. The n estimator is set to 50 by default. learning rate: This option is used to reduce each classifier's contribution. It is given a value of 1 by default.
SAMME or SAMME.R are two algorithms that can be used. Here we used SAMME.R as it uses probability estimations to update the additive model, whereas SAMME only uses classifications. And also the SAMME.R method converges faster than SAMME, resulting in lower test error and fewer boosting repetitions.
So, for the AdaBoost Machine learning model, we will evaluate the model by using repeated classified 5-fold cross-validation with the SAMME.R Algorithm.

Evaluation
Firstly, we build the AdaBoost model based on decision tree depth equal to 1 and cross-validation equal to 5 without calibration and tuning. Here we get an accuracy of the Adaboost model is 0.98 which is higher than some of the other models' performance. While validation accuracy scores are [0.98467815 0.98467815 0.98467815 0.98467815 0.98467803], our R-squared is only -0.016, which is not great in interpretability as we expected.

In this case, we take the balanced dataset from the previous oversampling step to rerun the Adaboost model. After we rerun the model, the R-squared decreased to -20.28 and test accuracy decreased to 0.67. Here we suspect two possible reasons, first, the model overfits to training data, or second, since AdaBoost minimizes an exponential loss, a mismatch between training and test loss could be the reason inducing the test accuracy decrease if we measure the performance using measures like 0-1 classification loss.

In order to tune the model and produce a more realistic output, we performed calibration to tune the model and calculate its performance. This time, we got improved AUC(0.69), precision(0.03), and F1 score(0.063) evaluations. The final model ended up with a more reliable confusion matrix compared to the before calibration one.

5.5 Network

Model description
Neural networks are a series of algorithms that mimic the operations of an animal's brain to recognize relationships in a set of data.

Hence, the structure of the neural network model is similar to that of the human brain. A "neuron" in a neural network is a mathematical function that collects and classifies information according to a specific architecture. The network bears a strong resemblance to statistical methods such as curve fitting and regression analysis.

A neural network contains 3 parts, an input layer, hidden layers, and an output layer. The input layer collects input patterns, such as the dimensions of each input record, the data type of each feature (numerical or categorical), and so on. The output layer returns classifications or signals to which the input patterns may map. As for hidden layers, they tune the inputting weights to minimize the margin of error.

Each layer is constructed by nodes (we call them 'neurons'), which are interconnected with nodes in previous and next layers. The function between connected layers can be recognized as a matrix, the data pass through these matrices and be activated by some activation functions.

The detailed structure of the model is manual, hence we need to find proper model construction with high accuracy and efficiency.

Model fitting
Firstly, we compare 2 model structures. The first model consists of 3 hidden layers, each has 128, 64 and 10 nodes respectively. And the second model has only 2 hidden layers with 128 and 32 nodes for the layers.

layer	**Output shape**	**Param #**
Input	(None,50)	
Dense 1	(None, 128)	6400
Activation 1	(None, 128)	
Dense 2	(None, 64)	8256
Activation 2	(None, 64)	
Dense 3	(None, 10)	650
Activation 3	(None, 10)	
Dense 4	(None, 1)	11
Output	(None, 1)	
total		15317

Table 3: Neural network model 1

After randomly split the data set and oversampling, we have 1,040,064 training data (with 520,032 fraud data and 520,032 not fraud data), 260,132 validation data (with 130,066 fraud data and

layer	**Output shape**	**Params #**
Input	(None,50)	
Dense 1	(None, 128)	6400
Activation 1	(None, 128)	
Dense 2	(None, 32)	8256
Activation 2	(None, 32)	
Dense 3	(None, 1)	650
Output	(None, 1)	
total		10561

Table 4: Neural network model 2

130,066 not fraud data) and 116,501 test data (with 114,604 fraud data and 1897 not fraud data).

Use the 'ReLU' function as an activation function in the hidden layers and the sigmoid function in the output layer. We choose ADAM as optimizer and binary cross entropy as loss function. Set batch size 128 and train the model for 30 epochs.

Seems like model 2 performs better over model 1, but considering the total training trend in figure 8, we found that the trend of validation loss and accuracy for model 1, or the deep model, is more stable than that for a shallow model.

Based on model 1's structure, we train the model on the training set for more epochs to reach higher accuracy.

Finally, our model has an accuracy of 72.74% on the test data.

6 Evaluation

6.1 Evaluation metrics

There are four main components that help to calculate accuracy, which are TP, TN, FP, FN. These components grant us the ability to explore other ML Model Evaluation Metrics. The formula for calculating accuracy is as follows:
TP represents the number of True Positives. This refers to the total number of observations that belong to the positive class and have been predicted correctly. TN represents the number of True Negatives. This is the total number of observations that belong to the negative class and have been predicted correctly.
FP is the number of False Positives. It is also known as a Type 1 Error. This is the total number of observations that have been predicted to belong to the positive class, but instead, actually, belong to the negative class.
FN is the number of False Negatives. It may be referred to as a Type 2 Error. This is the total number of observations that have been predicted to be a part of the negative class but instead belong to the positive class.

Precision
This refers to the proportion (total number) of all observations that have been predicted to belong to the positive class and are actually positive. The formula for Precision Evaluation Metric is as follows:

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

Recall
This is the proportion of observations predicted to belong to the positive class, that truly belongs to the positive class. It indirectly tells us the model's ability to randomly identify an observation that belongs to the positive class. The formula for Recall is as follows:

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

F1 Score
This is an averaging Evaluation Metric that is used to generate a ratio. The F1 Score is also known as the Harmonic Mean of the precision and recall Evaluation Metrics. This Evaluation Metric is a measure of overall correctness that our model has achieved in a positive prediction environment. Of all observations that our model has labeled as positive, how many of these observations are actually positive. The formula for the F1 Score Evaluation Metric is as follows:

$$\text{F1-Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

AUC
It stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).

Accuracy
Accuracy is also used as a statistical measure of how well a binary classification test correctly identifies or excludes a condition. The accuracy is the proportion of correct predictions (both true positives and true negatives) among the total number of cases examined. The formula for

quantifying binary accuracy is:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

Confusion matrix
Confusion matrix is a table layout that allows visualization of the performance of an algorithm. Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class, or vice versa – both variants are found in the literature.

6.2 Evaluation on test data

F1-score, accuracy, precision, recall, auc and recall-auc are shown in Table 4.

model	**f1 score**	**accuracy**	**precision**	**recall**	**auc**	**recall-auc**
Logistic Reg.(before fine-tuning)	0.0462	0.7080	0.0243	0.4616	0.5867	0.2471
Logistic Reg.(after fine-tuning)	0.0444	0.5516	0.0230	0.6800	0.6148	0.3539
Random Forest	0.0627	0.6964	0.0329	0.6627	0.6799	0.3504
LightGBM	0.1093	0.9518	0.0763	0.1928	0.5782	0.1407
Adaboost (before fine-tuning)	0.0629	0.6628	0.0329	0.7191	0.6905	0.3782
Adaboost (after fine-tuning)	0.0638	0.6702	0.0334	0.7142	0.6918	0.3761
Neural Network - 4 layers	0.0708	0.7274	0.0375	0.6378	0.6833	0.3406
Neural Network - 3 layers	0.0724	0.7358	0.0384	0.6331	0.6853	0.3387

Table 5: Evaluation of all models

Confusion matrix:
The confusion matrix of Logistic Regression model before fine-tuning is:

$$\begin{bmatrix} 108870 & 44083 \\ 1282 & 1099 \end{bmatrix},$$

Logistic Regression after fine-tuning:

$$\begin{bmatrix} 84056 & 68897 \\ 762 & 1619 \end{bmatrix},$$

confusion matrix of Random Forest:

$$\begin{bmatrix} 106603 & 46350 \\ 803 & 1578 \end{bmatrix},$$

confusion matrix of LightGBM:

$$\begin{bmatrix} 147394 & 5559 \\ 1922 & 459 \end{bmatrix},$$

confusion matrix of Adaboost before fine-tuning:

$$\begin{bmatrix} 101204 & 51684 \\ 687 & 1759 \end{bmatrix},$$

confusion matrix of Adaboost after fine-tuning:

$$\begin{bmatrix} 102352 & 50536 \\ 699 & 1747 \end{bmatrix},$$

confusion matrix of 4-layer neural network:

$$\begin{bmatrix} 83537 & 31067 \\ 687 & 1210 \end{bmatrix},$$

confusion matrix of 3-layer neural network:

$$\begin{bmatrix} 84517 & 30087 \\ 696 & 1201 \end{bmatrix}.$$

We can tell from the evaluation metrics and the confusion matrix that lightGBM has the highest accuracy on the test set. But we can also notice that it has low recall on the test set as well. Based on the confusion matrix we can conclude that it is because lightGBM have better prediction on the not fraud group, but cannot predict fraud transactions well. Thus, the high accuracy on test set cannot indicate that lightGBM is a good model in that scenario. Based on the model result.
In fraud detection, the main purpose is to figure out the fraud transactions. It is acceptable to have some missclassification on the non-fraud group. Thus, we can conclude that Adaboost is powerful in predicting the fraud transactions.

7 Conclusion

In the models we applied to fraud testing issues, due to the complexity and imbalance of our data set and real-life constraints, we only reached an accuracy of around 70% on test data.

Comparing these models, the lightGBM model's f1 score and precision have significant strength over other models but its recall is bad. That's, it is less likely to mistakenly classify a not fraud instant into a fraud. But concurrently, it will not detect most fraud situations.

For other models, they can judge fraud instances with higher accuracy but they easily classify not fraud records into fraud. What's worse, due to the imbalance of these 2 cases, the cases our models recognize as fraud are almost not fraud cases. The efficiency of dealing with those machine-tell fraud cases is low.

The limitation of models on this problem is clear, how can we improve it? Here are some suggestions:

- Data quality: there are 28 variables in our data, some are important, some make no sense, and most attributes are about the transactions and credit card, but not the customer. Information about customers can give us a clearer boundary between fraud and not a fraud.
- Oversampling and undersampling tricks: in our work, we only applied oversampling on training data to make fraud: not fraud 1:1. There are two possible approaches to improve this, one is combining oversampling and undersampling, and another is changing the proportion of fraud to not fraud data.
- Convert classification problem into regression: although our target label indicates a binary classification problem, consider the property of most classification models: they return a 'probability' in [0, 1], and then we give each instance a class that compares the probability against a threshold (usually 0.5). However, if we stop on the steps of probability output and skip the classification, and regard the probability as a possibility of fraud, then we are more likely to avoid misclassifying the not fraud situation into a fraud, but a 'warning' maybe.

8 Reference

.P, A. (2018, November 11). L1 and L2 regularization. Medium. Retrieved May 5, 2022, from https://medium.com/@aditya97p/l1-and-l2-regularization-237438a9caa6

Chaudhury, S. (2020, August 31). Tuning of adaboost with computational complexity. Medium. Retrieved May 5, 2022, from https://medium.com/@chaudhurysrijani/tuning-of-adaboost- with-computational-complexity-8727d01a9d20

L1 penalty and sparsity in logistic regression. scikit. (n.d.). Retrieved May 5, 2022, from https://scikit-learn.org/stable/auto_examples/linear_model/plot_logistic_l1_l2_sparsity.html

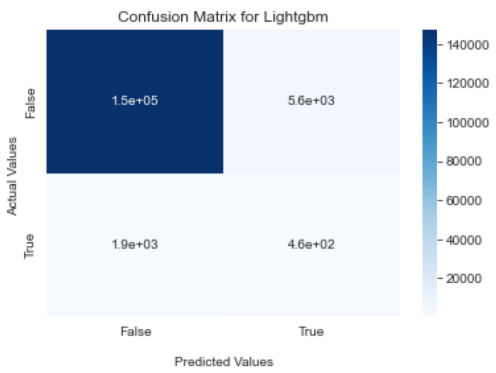
Sklearn.linear_model.logisticregression. scikit. (n.d.). Retrieved May 5, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.linear _model.LogisticRegression.htmlsklearn.lin

Sklearn.model_selection.GRIDSEARCHCV. scikit. (n.d.). Retrieved May 5, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

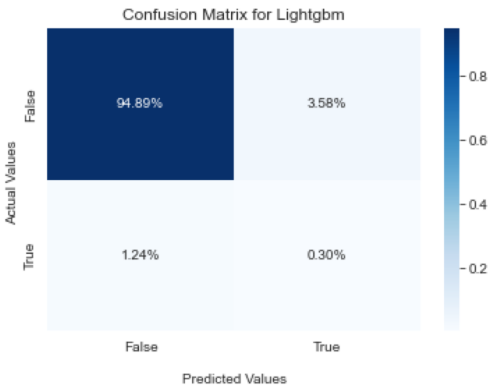
9 Appendix

	Missing number	Missing percentage
echoBuffer	786363	1.00
merchantCity	786363	1.00
recurringAuthInd	786363	1.00
posOnPremises	786363	1.00
merchantZip	786363	1.00
merchantState	786363	1.00
acqCountry	4562	0.005801
posEntryMode	4054	0.005155
merchantCountryCode	724	0.000921
transactionType	698	0.000888
posConditionCode	409	0.000520

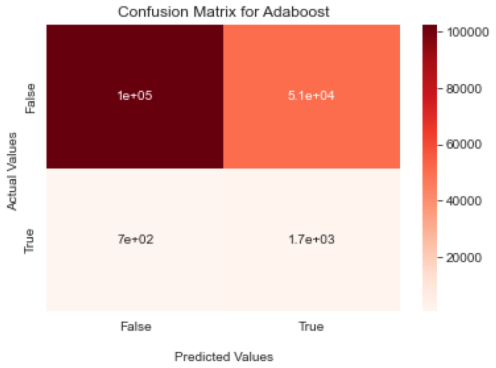
Table 6: Percentage of missing value



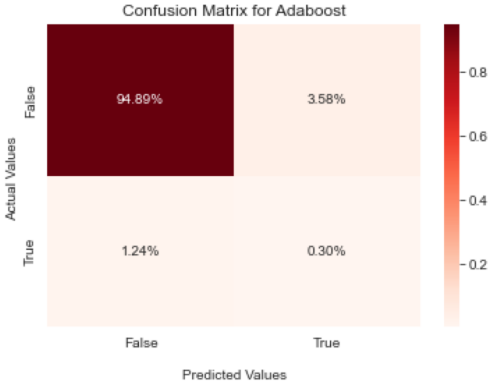
(a) Confusion matrix for LightGBM



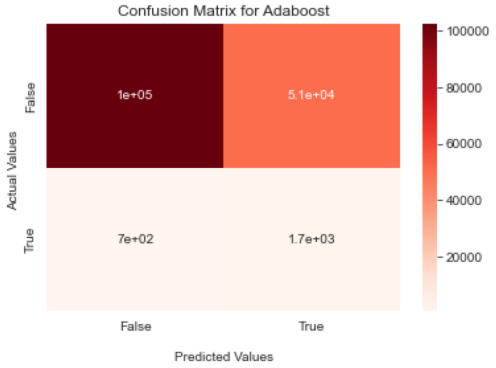
(b) Confusion matrix for LightGBM(pct)



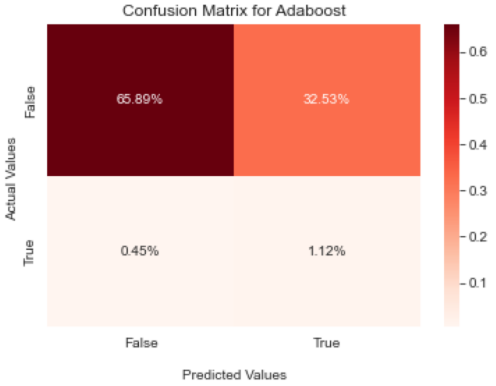
(c) Confusion matrix for Adaboost



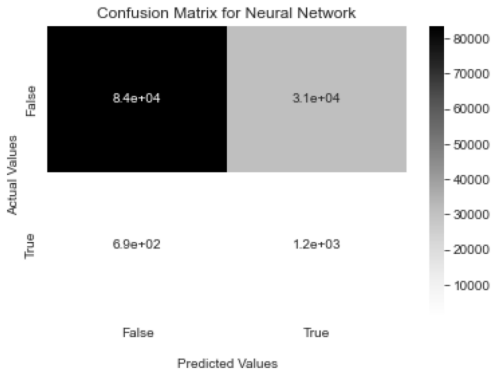
(d) Confusion matrix for Adaboost(pct)



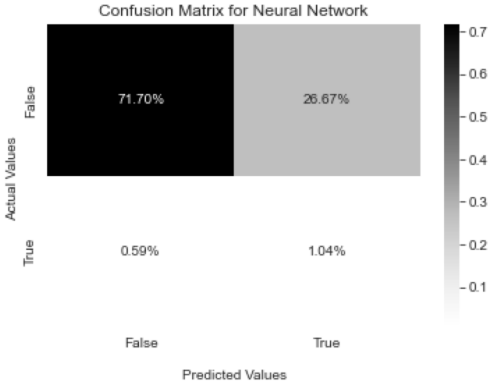
(e) Confusion matrix for Adaboost



(f) Confusion matrix for Adaboost(pct)



(g) Confusion matrix for Neural Network



(h) Confusion matrix for Neural Network(pct)

Figure 9: Confusion matrix for the models

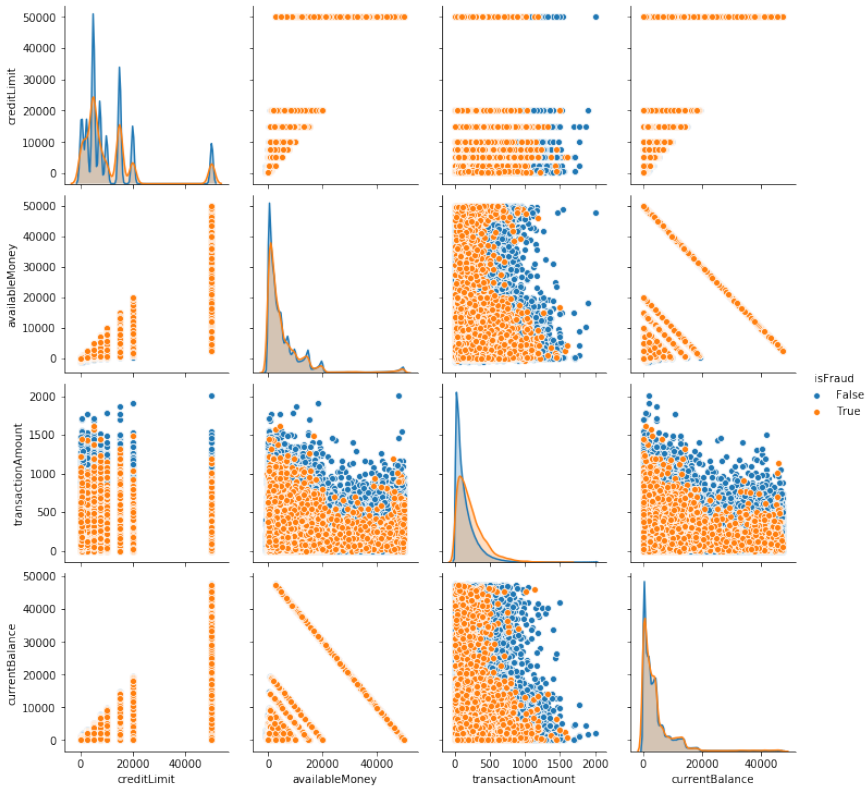


Figure 10: correlation plot with isFraud

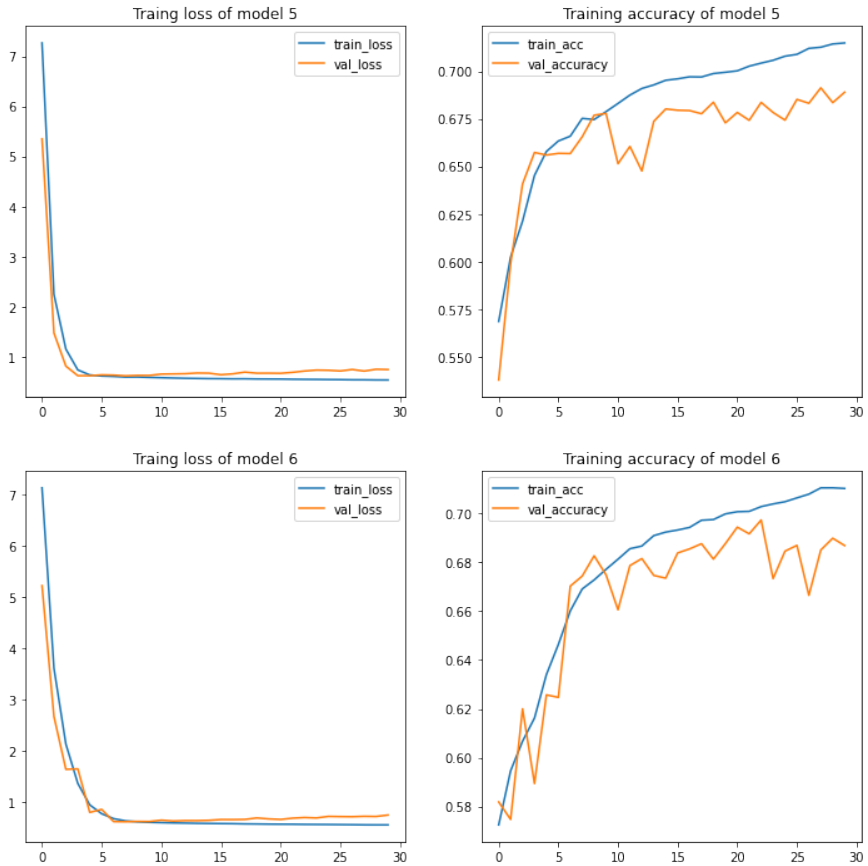


Figure 11: Training history of neural networks